University of Nottingham
Department of Psychology

From Soup To Nuts: Developing a Soar Cognitive Model of the Electronic Warfare Officer's Task

Gordon. D. Baxter

Working Paper: WP/R3BAIA005/014

Put into PDF May 2004, originally prepared 30 October 1997

Email: g.baxter@psych.york.ac.uk

Acknowledgements

Support for this work has been provided by the DRA, contract number 2024/004.

Table of Contents

Part 1: Introduction

1. Introduction

The purpose of this document is to describe the development of a model of the Electronic Warfare (EW) Officer's Task. In so doing it tries to satisfy one of the aims of the project (see Booth & Sheppard, 1996), which is to analyse, model and predict individual problem solving and decision making behaviour in a variety of command information system environments using a computational model of cognition, namely Soar (Laird, Newell, & Rosenbloom, 1987). The intention is that by integrating aspects of the current best practice from the field of cognitive modelling it should be possible to create an exemplar of how to develop cognitive models.

In general, little has been written about *how* to develop cognitive models. Most of the existing literature simply describes models that have already been implemented. There is a central hypothesis underlying this document that a cognitive model can be viewed as a computer program (or expert system). Taking this view suggests that the techniques available within software engineering (and artificial intelligence) for developing programs should be applicable to the development of cognitive models.

In addition to providing details of the development of the model, this document also attempts to provide some explanation for design decisions, in order to illustrate the sorts of problems that are liable to be encountered when developing Soar models. The document also provides some pointers to the organisation of the code into files, and suggests a possible file structure which may make it easier to trace problems when they arise.

The approach described here parallels some of the ideas described by Yost (1996) in connection with developing a model of an elevator configuration task using Soar/TAQL. The development (design and implementation) of a Soar model is essentially regarded as a three phase activity. As in all good engineering, however, the development is performed in an iterative manner, supporting backwards as well as forwards transitions between phases. Yost's final model was written using TAQL, but the final model here will be written using Soar 7. The three phases involved in the development of a Soar model are:

1. Development of a knowledge level description of the task.

2. Development of a Problem Space Computational Model level description of the task

3. Development of the Soar 7 model

The transformations from the higher to lower levels are performed in such a way as to try to preserve the structure of the higher level model(s).

The aim of the first phase is to try and identify all the discrete types of task knowledge that can be brought to bear whilst performing the task. One way of doing this is to use a static analysis method, such as Hierarchical Task Analysis (HTA) (Annett, Duncan, Stammers, & Gray, 1971). The main disadvantages of methods such as HTA is that they tend to be prescriptive, rather than descriptive. It does, however, enable a preliminary attempt to be made at identifying the sort of knowledge that is required to do the task.

In order to ascertain the knowledge that *is* used, rather than the knowledge that *should* be used, the knowledge identified by the HTA will be supplemented and modified by knowledge elicited from an expert. The strategies utilised by the expert in doing the task will also be used to determine how the model should apply the knowledge that it has.

In parallel with the development of the model, an experiment is being conducted using the same task simulation. Any results that become available from the experimental EW study may be incorporated into the model, subject to the provisos that there is sufficient time available to add in the extra

features, and that the intention of the project is **not** to create normative models of expertise in the task.

In addition to the domain and task knowledge that will be identified by the HTA, the model also needs to encapsulate knowledge about how to utilise computer interfaces. This sort of knowledge is task-independent, and will include aspects such as how to press keys, how to move the mouse, what to do with mouse buttons, and so on.

2. EW Task Description

The task and the details of the experiment are described at length in the EW task manual (Chapman, Ritter, & Baumann 1996). A short overview of the task is provided below in order to facilitate understanding of the HTA that follows it.

The basic aim of the task is to prevent own ship from being hit by incoming missiles. The only way to decoy missiles is by firing chaff, using the onboard jammer, and by manoeuvring the ships heading and speed as appropriate. The application task consists of a number of scenarios that are run using the OOPSDG simulation (Ramsay, 1995). The task is performed using an interface developed using SL-GMS (Sherrill-Lubinski Corporation, 1994).

The task is presented in the experiment using a dual task paradigm. In the real world, the EW Officer would not always sit down at the display with the certainty that there would be some incoming missiles to deal with in the immediate future. To try and take account of the fact that there is a certain amount of vigilance involved in the real task, experimental subjects are therefore also asked to perform a tracking task. The main task is still to prevent own ship from being hit by any threats that may appear whilst performing the secondary task. The secondary task involves tracking the movements of a helicopter as it travels across the display, trying to keep own ship at a fixed distance from the helicopter. The experimental task has limited surface similarity with the task of a real EW Officer: the secondary task would normally be handled by other members of the ship's crew, for example.

The only means available to deal with incoming missiles are chaff rounds, which are in limited supply, and radar jamming, although only one missile can be jammed by any one jammer at a time. When an incoming missile is detected, the EW Officer has to decide on which countermeasures to use to decoy the missile, and how to manoeuvre own ship to minimise the chances of being hit by the missile. Since the jammer uses an active radar signal, it makes own ship visible to other radar, so permission has to be requested before any of the jammers can be used. In order to simplify the task somewhat, only one type of missile is used: a heat-seeking anti-radiation missile.

The task has to be performed using a single head system (i.e., only one display screen), whereas normally a second display screen would show threat information in a tabular form. The amount of information that is available to the operator is therefore limited to that shown in Figure 1. The only information that is directly available about the objects shown on the radar part of the display—the Plan Position Indicator, or PPI—is whether they are friendly or hostile, and the track number associated with each object. Friendly vessels are shown in blue, whilst hostile ones are shown in red. The only other information that is available from the PPI is the approximate heading of own ship, shown as a dashed white line. For hostile objects, a dashed red line is shown for a missile that is in search mode, and this becomes a solid line when the missile is locked onto own ship.

There is more information about own ship, which is shown at the bottom of the display screen (see Figure 1). These controls show the current heading, and speed of own ship, together with the status of the various countermeasures. They also allow the operator to activate the countermeasures.

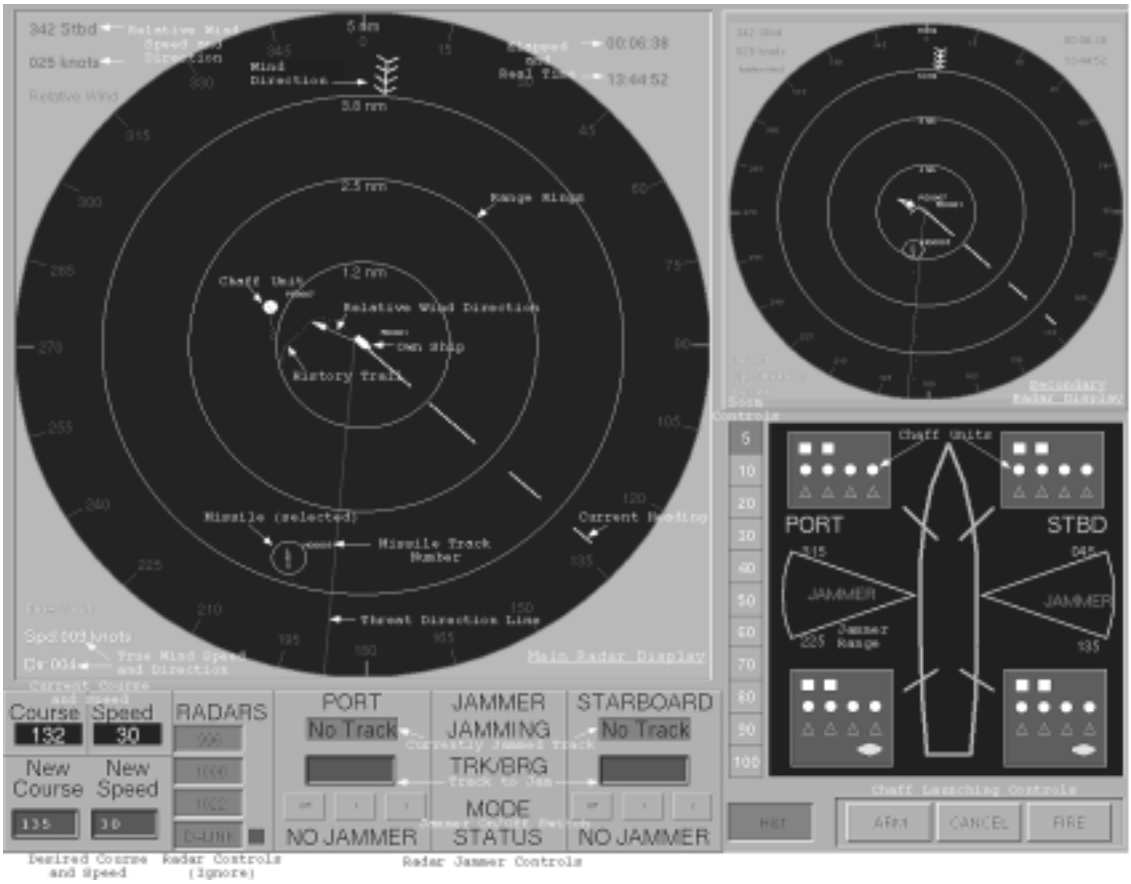

Figure 1. The EW Simulation Display.

Part 2: The Knowledge Level Model

3. Developing the Knowledge Level Model

An HTA is presented below to show how the task can be decomposed into operations and plans. It is assumed that the task is perceived as a single task, which has two components to it. As with any HTA, however, the precise interleaving of the individual task operations cannot be specified in an a priori manner, since it will heavily depend on situational factors, such as whether there are any missiles currently heading towards the ship, how many there are, how far away they are, whether chaff has already been launched, and so on.

3.1 Iteratively Refining the HTA

The HTA is presented below in a tabular format similar to the one used by Diaper (1989). The stopping condition used to terminate each branch of the analysis on the first pass is the point where the EW Officer has to interact with the application task interface. Actions such as check for missiles are thus used as a stopping condition, since it involves physically moving the eye in order to look around the display screen. By setting a stopping condition at a relatively high level of abstraction, the need for sophistication in the model is reduced.

Assumptions about the task have also been made in order to simplify the construction of the model. So, for example, chaff always arm, fire, and bloom successfully. The net effect is that the need to continually look for feedback when such actions are performed is alleviated. Extra detail and complexity can be added into the model at a later date, as and when required.

The final version of the HTA has been subjected to two major levels of refinement, which could have been carried out in any order. The first involves taking account of the physical interactions that the model has to perform in order to do the task (see Appendix A for more details). Carrying out this refinement allows the appropriate operations for performing sub-tasks such as searching for missiles to be fully laid out. The knowledge needed to perform these operations is task independent, and is included here for completeness. Refining the task in this way increases the depth of the HTA and therefore correspondingly increase its complexity (in terms of the number of individual actions needed to perform a particular sub-task).

The second major refinement is based on the results of a Knowledge Elicitation exercise carried out on a subject matter expert. The main effect of this refinement is to modify the strategies adopted by the model in order to perform the task. More details of the Knowledge Elicitation exercise are provided in Appendix B.

The most obvious effects of having to make successive refinements to the HTA is that the models at the Problem Space Computation Model (PSCM) level and the ensuing implementation also require comparable modifications in order to ensure that they comply with the higher level analyses. The project will therefore produce a set of Soar models, one for each of the levels of refinement. There are some parallels here with the ideas of evolutionary prototyping, a technique used in Software Engineering. The first prototype model is extended and refined to provide the final model, rather than simply being discarded, as happens in rapid prototyping.

Table 1. Tabular Format Hierarchical Task Analysis for the EW Task.

Superordinate	Task Analysis–operations–plans	Notes
0.	EW Task *Plan 0: 1–2–If end of watch–EXIT; If not end of watch, start the cycle from 1 again.* 1. Handle missiles// 2. Track helicopter//	Task is to avoid hostile missiles, and track the flight path of the helicopter.
1.	Handle missiles *Plan 1: 1–If no missiles present–EXIT; if missiles detected–2, then cycle from 1* 1. Check for missiles// 2. Decoy missiles//	Checking for missiles involves scanning the display screen. Is multiple missile strategy different, rather than do each one in turn?
2.	Track helicopter *Plan 2: 1–2 in any order then EXIT* 1. Control ship heading// 2. Control ship distance from helicopter//	In reality the expert circled the ship round by changing heading, rather than modify the speed of own ship.
1.1	Check for missiles *Plan 1.1: 1–2–EXIT* 1. Check for new missiles// 2. Check old missiles//	Once a missile has been successfully decoyed, it is not checked again.
1.2	Decoy missiles *Plan 1.2: 1–2–3–4–5–6–If chaff fails to bloom repeat 4 then 6 until chaff blooms–If more missiles cycle from 1 otherwise EXIT* 1. Get missile details// 2. Select missile 3. Get relative wind heading// 4. Deploy chaff// 5. Deploy jammer// 6. Check for chaff bloom//	Normally, relative speed would be used to select a missile to decoy first. In this task, however, all missiles travel at the same speed. Always use chaff *and* the jammer, if permission is granted.

1.1.1	Check for new missiles *Plan 1.1.1: For every undecoyed missile 1–2–then Exit.* 1. Get missile track number// 2. Add missile track number to undecoyed list//	This is basically a perceptual sub-task.
1.1.2	Check old missiles *Plan 1.1.2: 1–If a change in missile heading is detected 2–EXIT; 3–If missile distance is less than ten miles 4; EXIT* 1. Check missile heading// 2. Mark missile as successfully decoyed// 3. Check missile distance// 4. Deactivate jammer//	Only check missiles which are not marked as successfully decoyed. A change in missile heading towards the chaff indicates a success, and can be picked up from the track.
2.1	Control ship heading *Plan 2.1: 1–2–If ship heading is same as helicopter bearing–EXIT; If ship heading is different from helicopter bearing–3–EXIT* 1. Get helicopter bearing// 2. Read ship heading// 3. Change ship heading//	The approximate helicopter bearing is obtained by reading it off the PPI.
2.2	Control ship distance from helicopter *Plan 2.2: 1–If helicopter distance greater than threshold–2–EXIT; If helicopter distance less than threshold–3–EXIT* 1. Estimate ship separation// 2. Increase own ship speed// 3. Reduce own ship speed//	Changing ship speed involves interacting with the interface.
1.2.1	Get missile details *Plan 1.2.1: 1–2–3–EXIT* 1. Get missile speed// 2. Get missile distance// 3. Get missile bearing//	Missile speed is fixed. Relative missile speed is the most important feature.

1.2.3	Deploy chaff *Plan 1.2.3: 1–2–3–4–5–6–7–EXIT* 1. Calculate escape route// 2. Select chaff dispenser// 3. Select chaff rounds// 4. Arm chaff rounds// 5. Fire chaff rounds// 6. Follow escape route// 7. Refine escape route//	The strategy is to fire chaff and turn the ship to minimise radar signature and head off to maximise separation between the ship and the chaff.
1.2.4	Deploy jammer *Plan 1.2.4: 1–2–If jammer in use then EXIT; 3–4–EXIT* 1. Select jammer// 2. Check if selected jammer in use 3. Request permission to use jammer 4. Activate jammer//	The jammer only works on one missile, and only functions across a 90° arc so the ship should not be turned when the jammer is used.
2.1.2	Change ship heading *Plan 2.1.2: 1–2–3–EXIT* 1. Find helicopter bearing// 2. Click in New Course box// 3. Enter helicopter bearing value//	
2.2.2	Increase own ship speed *Plan 2.2.2: 1–2–3–EXIT* 1. Calculate increased speed// 2. Click in New Speed box// 3. Enter new speed//	
2.2.3	Reduce own ship speed *Plan 2.2.3: 1–2–3–EXIT* 1. Calculate reduced speed// 2. Click in New Speed box// 3. Enter new speed//	

1.2.1.1	Get missile distance *Plan 1.2.1.1: If PPI scale is too coarse–1–2–3–EXIT* 1. Calculate new range for PPI// 2. Click on new range value// 3. Estimate missile distance//	Use the clickable range buttons to zoom in and out of the PPI.
1.2.3.1	Calculate escape route *Plan 1.2.3.1: 1–2 in any order then EXIT* 1. Calculate new ship speed// 2. Calculate new ship heading//	
1.2.3.6	Follow escape route *Plan 1.2.3.6: 1–If a jammed missile would fall out of jammer arc–2; 3–4–EXIT* 1. Check jammed missiles// 2. Deactivate jammer// 3. Set new ship heading// 4. Set new ship speed//	Turn off the jammer if jammed missile would finish out of the arc of the jammer.
1.2.3.7	Refine escape route *Plan 1.2.3.7: 1–If new heading is the same as the current course–EXIT; otherwise 2–3–EXIT* 1. Calculate new ship heading// 2. Click in New Course box// 3. Enter new heading//	In general the heading can be tweaked to a more optimal one. Speed is usually already set to top speed.
1.2.3.6.1	Set new ship heading *Plan 1.2.3.6.1: 1–2–3–EXIT* 1. Calculate new heading// 2. Click in New Course box// 3. Enter new heading//	

1.2.3.6.2	Set new ship speed *Plan 1.2.3.6.2: 1–2–3–EXIT* 1. Calculate new speed// 2. Click in New Speed box// 3. Enter new speed//	New speed will depend on complexity of the scenario, but will usually be the maximum speed.

3.2 Task Specific Knowledge

The following elements of task knowledge can be explicitly identified from the tabular HTA shown above, with the exception of radar knowledge. Although there are facilities available for making use of radars in the task interface, they are not allowed in this task. Radar knowledge is included in the list, however, for the sake of completeness, since radar is part of a ship's defences against incoming threats.

1. Own ship speed
2. Own ship heading
3. Relative wind
4. True wind
5. Decoy knowledge
6. Radar knowledge
7. Chaff knowledge
8. Jammer knowledge
9. Missile knowledge

One very important piece of knowledge is missing from this list: display knowledge. Implicit in the task is an awareness of what the task specific display screen can be used to accomplish.

The HTA has already shown how the different elements of knowledge can be applied. In the rest of the section, the individual elements will be described more fully, considering aspects such as whether they can be modified, and how to access the knowledge. Some of the knowledge is strategic, and will remain within the cognitive model, whilst other parts of it are more tactical, and may be held in the world: this accords with the notions of knowledge in the head, and knowledge in the world (Norman, 1988). Of particular use are the preconditions that are needed to apply a specific piece of knowledge. The granularity of the different types of knowledge is also important, in terms of whether all of one type of knowledge is accessible at once, or it has to be assembled on the fly.

3.2.1 Display knowledge

Display knowledge does not emerge directly from the HTA, although it does form the immediate task environment. This knowledge is task specific, since it relates to the particular display used on the EW Task. Display knowledge provides a link between the task specific knowledge and the task independent knowledge. The task independent knowledge is applied to bring the results of using the task specific knowledge to bear on the task. In order to interact with the application task the user has to perceive the information that is currently displayed, and act upon it.

The display screen reflects the current state of the task, and as such provides a repository for knowledge in the world, in a sort of scratch pad. Display knowledge is thus intrinsically linked to many of the other types of knowledge used in the task. Thus, for example, if the user wishes to zoom in or out of the main PPI, there are buttons available to accomplish this. This is the main part of the display knowledge that is not directly connected with aspects of the task. Zooming is really an artificial aid, whereas all of the other elements of task specific knowledge would still be required if there was no display screen available.

Zooming in and out of the PPI requires the use of eye knowledge, in order to move the model's eye around to find the command button labelled with the desired radius value. Once this has been found, mouse knowledge has to be utilised in order to move the mouse pointer over the chosen command button, and then the mouse button has to be clicked.

The small PPI display is not used, since the scale it offers is not of any great use. When checking for missiles the main PPI is zoomed out to 100 miles. When missiles are being actively tracked and decoyed, efforts are made to keep the missile close to the outer range ring, by zooming in as appropriate.

3.2.2 Own ship speed

Own ship speed can be read from the display screen and modified by the model. There is a time lag involved between the entering of a new value, and the ship actually reaching the new speed. The speed also fluctuates as the ship moves, reflecting the influence of extrenal factros such as the prevailing wind conditions.

There are two separate situations where own ship speed needs to be changed. The first is during contact with a missile, when the speed is changed to maximise separation between the chaff decoy and own ship. The second situation is where the distance between own ship and the helicopter which is being tracked changes outside the desired limits, speed may need to be adjusted in order to regain the required separation.

When the need to alter speed has been identified, a new speed has to be calculated. In the case where the change is part of the missile avoidance procedure, the general rule of thumb is to increase to top speed (30 knots) in order to maximise separation between own ship and the chaff bloom. In more complex scenarios, where multiple missiles are involved, it may be necessary to adopt a different strategy for selecting the new speed.

In the case where own ship is trying to regain station with the helicopter, the new speed has to be calculated based on the current bearing of the helicopter—its direction from the ship—since the current heading of the helicopter—its direction of travel—is not available. The new value should be calculated on the basis of regaining station within a reasonable period of time. Unless the distance between own ship and the helicopter has become excessively large, it is not sensible to ramp up to top speed, due to the amount of time (and therefore distance) required to slow the ship down. Normally when performing the task, the expert did not alter speed to regain station with the helicopter, but instead achieved the desired effect by circling round (changing the heading as appropriate).

Once the desired new speed has been calculated, the speed of own ship can be modified. The first step requires that a new value be typed into the New Speed field on the display (any existing value in that field has to be deleted first). There is then a finite time delay as the ship accelerates or decelerates to reach the desired speed. During the acceleration to the new speed, the value shown in the Speed field on the display will update at a fixed rate. The maximum speed of own ship is 30 knots.

3.2.3 Own ship heading

Own ship heading can be changed under two sets of conditions. The first is during contact with a missile, where the missile is to be decoyed using chaff. In this case the change of heading is the first action to be performed after the chaff has been fired. The second situation is where the ship has to be turned in order to track the flight of the helicopter. As noted above, the expert changed heading to regain station with the helicopter, rather than speed. Instead of slowing down, he circled round, with the radius of the circle dependent on the distance that he was ahead of the helicopter.

The current value of own ship heading can be read directly from the display screen. The new value of the heading needs to be calculated on the basis of the current situation. The calculation of the desired

value requires the use of modulo 360 arithmetic to cater for changes in heading that cross the 360° threshold (e.g., a heading 40° east of 340° equates to a desired heading of 20°).

During contact with a missile, the ship has to be turned to a heading such that it minimises the radar profile it presents to the incoming missile. The basic shape of the radar profile can be approximated by a Swiss cross (all arms of the cross are equal); the accurate representation is more complex, and bears some resemblance to an heraldic bird. Changing the heading is a two phase operation. In the first instance a coarse change of heading is performed, with the aim of simply trying to buy some time to make good the decoying of the missile. Once the chaff has successfully bloomed, the ship's heading is then subject to a fine adjustment to a more optimal course.

When simply trying to maintain (or regain) station with the helicopter, the new heading value is calculated based on the current bearing of the helicopter. In the initial version of the model, station will be regained by adjusting speed and heading in tandem, rather than using the more complex strategy of circling round at constant speed.

Once the desired new heading has been calculated, the heading can be modified. The first step is to enter a new value, by typing it into the New Course field on the display (after deleting any existing value). The value in the Course field will then change at a fixed rate (as the ship turns) until the desired heading is attained.

3.2.4 Relative wind

Relative wind is the resultant wind after taking account of the true wind, and the speed and heading of own ship. The relative wind is shown in two formats on the PPI (see Figure 1). There is a short white arrow which points from own ship in the direction that the relative wind is blowing. This indicator shows the general direction of the relative wind, and is not labelled with a numeric value. The numeric value of the relative wind is shown in the top left hand corner of the PPI. It consists of a speed, and a direction that is relative to own ship.

The relative wind information is consulted as part of the process of decoying a missile. In order to decoy a missile, the separation between chaff and own ship needs to be made sufficiently large so as to force the missile to head for the chaff. The current relative wind provides an indication of how quickly own ship and the chaff bloom will separate. The aim is to try to maximise the difference between own ship and the chaff, whilst simultaneously making sure that own ship does not become an easy target for any other missiles.

Relative wind information is critical to the task. In order to maximise the separation between own ship and the chaff, the effect that is being looked for is large pockets of relative wind blowing across the ship's deck, rather than any particular wind direction.

3.2.5 True wind

True wind is depicted in two formats on the PPI. A graphical indicator is shown close to the outer range ring, in the form of a number of white chevrons pointing in the direction that the wind is coming from. The actual values (speed, and direction) are shown at the bottom left hand corner of the PPI (see Figure 1).

Under the conditions being modelled, true wind is not explicitly used. It is normally only used implicitly when choosing between different ship headings whilst trying to decoy chaff.

3.2.6 Decoy knowledge

Decoy knowledge is strategic knowledge, since it is based around plans of action for how to cope with incoming hostile missiles. There are a number of capabilities that are required in the deployment of decoy knowledge. The ability to detect a hostile missile, which requires the use of eye knowledge

There is a general strategy which can be employed to decoy incoming missiles. This is based on a relatively fixed sequence of operations that can be used whenever a missile is detected. The first step involves selecting the missile to be decoyed. This decision is based on the relative speed of the missiles, with the missile which would contact first being selected first. In the EW Task this is not an issue, since *all* missiles travel at the same speed. Next, a decision is made about how it is to be decoyed: either by using chaff, or jamming it, or both. Generally, the strategy is to use whatever means available, so the relevant jammer (if available) *and* chaff will be used. Once this decision has been taken, the relevant course of action is chosen based on chaff knowledge, and jammer knowledge.

If a missile locks on to the ship, priority is given to decoying that missile. It is possible to break lock by setting up a chaff bloom with a larger radar signature than that of the ship, and moving out of the missile's scanning range. When lock is broken, the missile returns to scan mode.

The strategy for dealing with multiple missiles may differ from the strategy for dealing with single missiles, depending on the particular situation. The choice of strategy will mainly depend on the number of missiles, their bearing, and their distance. It may be possible, for example, to deal with two missiles coming from approximately the same distance and direction by simple using the strategy for dealing with a single missile: deploying chaff, and heading off at an angle and speed that will maximise separation between own ship and the missiles whilst presenting a reduced radar profile to the missile.

3.2.7 Radar knowledge

Although there is a section of the display screen allocated to radars, under the task being modelled, radar knowledge is not required, because the radars are not used.

3.2.8 Chaff knowledge

The only countermeasures that are available for decoying missiles are through the use of chaff, and the use of radar jamming. There are two sets of chaff dispensers, one on the port side, and one on the starboard side of the ship. Each set consists of a pair of units, located fore and aft on the ship. The number of rounds of chaff available is limited, so chaff should be deployed in a relatively conservative manner, rather than simply sowing blankets of chaff to hide behind whilst trying to escape from a missile..

There is a conflict between using chaff and using the jammer, since if the jammer is currently active, care is needed when calculating how far to turn the ship. If the size of the turn is too great, the missile that was being jammed may end up outside of the arc of the jammer, and hence can start homing in on the ship once more. The solution to this is to turn the jammer off.

Prior to firing chaff, a decision has to be made about which set of dispensers to use. The choice depends on the current heading of the ship, and the relative wind direction, and the target. The target object is selected by clicking on it which causes it to be highlighted on the display, where it gets surrounded by a white circle.

Once a target has been selected, there is a fixed sequence of actions to be performed in order to fire chaff: a round (or a number of rounds) has to be selected, then armed, and then fired. Each of these actions is accomplished by clicking the mouse on the appropriate interface object. The chaff then takes a fixed time to bloom (around 15 seconds), at which point it appears on the PPI as a filled white circle. It is possible to have faulty chaff rounds, however, which may not bloom, so the display has to be checked to make sure that each round has worked correctly. If a chaff round fails to bloom, another round is normally fired. After blooming, the chaff decays over time, and eventually disappears (after around 200 seconds).

If a missile fails to be decoyed by a round of chaff, and that missile is closing in on own ship, another round of chaff is fired. The aim is to fire it close to the original round, in order to increase the size of

the radar signature of the chaff, and reduce the relative size of the radar signature of own ship. The fact that the chaff takes 15 seconds to bloom is taken into account as part of the decision on whether to fire extra rounds to try and decoy a particular missile.

Chaff knowledge is employed when an incoming missile is detected as being within a range that makes it susceptible to being decoyed. Normally the strategy is to fire chaff early with a view to buying time to make good an escape. In order to estimate the distance between own ship and an incoming hostile missile, it is necessary to zoom in and out of the main PPI. The general rule is to try to use the level of zoom of the PPI which keeps the missile as close as possible to the outer range ring.

3.2.9 Jammer knowledge

The ship has two jammers onboard, one on the port side, and one on the starboard side. Each of the jammers operates in an arc between 45° and 135° on its own side of the ship. The jammer interferes with the radar detection capability of the missile. When not actively being used to decoy a missile, the jammers should be switched off. Each of the jammers jammer can only be used to jam one missile at a time.

Using the jammer increases the ship's detectability, so permission has to be sought before turning any of the jammers on. A jammer will only be activated when an incoming missile falls within the operating arc of the jammer. In order for a jammer to be activated, it must currently be deactivated. Once a jammed missile has been successfully killed, the jammer should be deactivated.

The basic strategy is to always try to use the jammer. This can be stymied by the jammer already being in use, or permission to use it being rejected. The underlying idea is to use all possible resources to decoy the missile.

The missiles in the EW Task scenarios have a home on jam capability, which allows them to home in on the jammer once they get within a certain range of the jammer. In order to counteract this, the jammer has to be deactivated when the jammed missile comes within a distance of about 10 miles of own ship.

3.2.10 Missile knowledge

A case could be made for having missile knowledge active all the time. In more complex scenarios, however, this knowledge would be partitioned into separate segments, one for each of the different missile types. In order to remain congruent with this idea, missile knowledge is activated when an incoming hostile missile is detected. For each of the missiles detected, the missile knowledge leads to the activation of the decoy knowledge in order to deal with the missile.

Under the operating conditions being modelled, there is only one type of missile. This is radar seeking missiles, which actively seek out targets with a radar reflection, and then home in on them. The missiles travel at a fixed speed, and can be decoyed by jamming, or by chaff. The missiles operate in two modes. In scan mode they operate using a wide search arc to seek out any objects with a large radar reflection. Once they have selected a target they move into lock on mode, where they use a much narrower search beam, and try to track the target.

Missiles, and all other hostile vessels are shown in red on the PPI (friendly vessels are depicted in blue). When a hostile missile is in scan mode it has a track associated with it which consists of a single red dashed line. When the missile moves into locked on mode, the dashed line becomes solid.

3.3 Task Independent Knowledge

In addition to the task specific knowledge elements identified above, there are a number of other (more or less) task independent elements of knowledge that people use when doing the task. These

elements of knowledge relate to interacting with the interface. If the model is to exhibit levels of task performance that approach human performance, the model needs to have the capability to interact with the interface in a similar manner to the way that a real user would (Baxter & Ritter, 1996b). The model therefore has to have the following available to it:

1. Mouse knowledge
2. Eye knowledge
3. Keyboard knowledge

Hand-eye co-ordination emerges naturally out of looking for feedback on a particular action.

3.3.1 Mouse knowledge

This is knowledge about how to use the mouse. It covers details such as how to click the mouse button (only the left mouse button is used), how to move the mouse, how to press the mouse button (and hold it down), and how to release the mouse button. These are discrete actions, encapsulated as discrete pieces of knowledge.

The knowledge about clicking the mouse button is needed to accomplish the following actions:

- Entering some new data into a field on the display (new heading, or new speed).
- Selecting an object on the PPI.
- Selecting a chaff round in preparation for firing.
- Arming, or firing chaff, using the command buttons.
- Selecting a jammer, using the command buttons.

3.3.2 Eye knowledge

The model has a coarse cyclopean eye available to it, which allows it to perceive what is on the display. The eye is described in more detail elsewhere (Baxter & Ritter, 1996a; 1996b). The model moves the eye around the screen, as appropriate, to look at the different objects that are currently displayed. The two basic pieces of knowledge about the eye which the model has are: how to get it to move, and how to fixate on what is currently projected onto the fovea of the eye. The information perceived by the model when it fixates is used in deciding where the eye should look next.

3.3.3 Keyboard knowledge

This is essentially knowledge about typing data values into fields on the display. The majority of the application task interface is mouse driven, rather than keyboard driven. The only typing that the model has to be able to do is to enter numeric values into a field, and to delete existing data from a field.

The knowledge about when to type in data values at the keyboard is utilised in the following cases:

- When the model wishes to enter data into a field, and that field is not empty, so the existing data has to be deleted.
- When the model wants to enter a new heading value into the New Course field.
- When the model wants to enter a new speed value into the New Speed field.

As a prerequisite for typing at the keyboard, the field that is to be updated has to have the current focus, i.e., has to be capable of accepting input at that time. In order to get focus the model may have to click the mouse button whilst the mouse pointer is currently located somewhere within the typing field.

Part 3 : The Problem Space Computational Model

4. Developing the PSCM-level Model

The PSCM level model partitions the knowledge identified in the Knowledge Level model into the different PSCM components: problem spaces; tasks (or goals); states; and operators. Although the precise workings of the PSCM have been revised somewhat since Yost's (1996) model was originally developed, the basic components remain unchanged.

The PSCM defines the knowledge level model in terms of structures that can be implemented by the Soar architecture. The basic components of the PSCM are:

- Problem Spaces: An organisational mechanism which is used to group together states and operators, and the knowledge that connects them.
- Tasks: Particular problems to be solved, or goals to be attained.
- States: Collections of objects which describe the current state of the task. Goals are simply desired target states.
- Operators: The mechanism for manipulating and transforming states.

The PSCM's knowledge roles are related to its components:

- determining an initial state
- proposing operators
- selecting operators
- applying operators
- recognising desired (goal) states

During each step in the task, the PSCM tries to bring to bear all the available knowledge for the current role. In cases where there is a lack of knowledge available to uniquely specify a decision, the PSCM automatically generates a new sub-task to remove the problem by gathering sufficient knowledge.

In more recent versions of Soar, the PSCM has been changed in such a way that the importance of the problem space as a construct has been reduced. The problem space is now simply an attribute of the state, and provides a means of grouping states together. The state is thus now a hybrid problem space/state construct. The net effect is that a change of problem space is now represented by a change of state with a corresponding change being made to the problem space attribute, although the latter has to be explicitly implemented by the modeller.

Only one operator can be applied to a state at any one time. Operators are used to define the central structure of the state, and when an operator is applied, any changes it makes to the state are permanent (unless they are removed by another operator). In addition to this method of modifying the state, there is also a method called state elaboration. Elaboration modifies the state by adding extra objects to it. Essentially elaboration determines that if some particular collection of objects is part of the state, then some other particular collection of objects should also be part of the state. The changes made by elaborations are not permanent: if the original collection of objects changes, then the objects added by that elaboration are removed from the state.

The central problem in defining the PSCM level model is to identify the appropriate states, operators and problem spaces that are needed to do the task. The structure of the PSCM level model is determined by a hierarchy of problem spaces and operators which perform transitions between states. State transitions are accompanied by a change in problem space when the new state does not form part of the current problem space. A Soar model of a particular task can therefore be conceived of as a finite state machine for that task. Operators effect the transformation between states, whereas elaborations simply modify the current state (in a non-permanent way).

There are additional complexities which arise when trying to model performance of real world tasks, where the goal of the task per se is ill defined. The goal of the EW task, for example is to avoid being hit by missiles, but this is an open-ended task since it is impossible to predict exactly when there will not be any other missiles to decoy. The difficulty lies in trying to define what the desired state should be. Fortunately, there are several sub-tasks which can be more precisely defined. It is possible, for example, to determine when one particular missile has been successfully decoyed, because the track for that missile will disappear from the PPI display. Even in this case, however, the definition of the desired state has been simplified, since there is a relatively long time lapse between a missile being successfully decoyed and its track disappearing from the display.

The PSCM components for the EW Task are identified below by filling out the PSCM knowledge roles for each of the problem spaces. It is worth noting at this point that within Soar, expert behaviour occurs at the top level problem space, consisting of a sequence of operator applications. In tasks such as the EW Task, the operators will normally be experts, having undergone many hours of training before doing the task for real, and will be briefed about what sorts of things to expect beforehand.

For the purposes of the model multiple problem spaces will be used, since the model will start out with a basic level of knowledge, and then will learn to do the task through practice, using Soar's learning mechanism, chunking. There are some important consequences of this decision, due to the constraints imposed by the Soar architecture.

In Soar, operators are implemented as three separate (groups of) productions: operator proposal rules, operator application rules, and operator termination rules. If a model has to learn how to apply a particular operator it has to use the Soar chunking mechanism, since this is the only learning mechanism in Soar. In order to create chunks, the model must reach an impasse when the operator is proposed, since chunks are learned when impasses are resolved. The operator will then be applied in a lower level problem space, where Soar will build a chunk across this particular problem solving episode, as it works out how to apply the operator.

Some care is needed in deciding exactly how to identify the PSCM components from the HTA. A useful general rule of thumb, which has been adopted here, is that each of the levels in the HTA constitute a problem space, and each of the operations which requires further decomposition is an operator which leads to a change of problem space. Within Soar, however, the only way to change problem spaces is via impasses, so these impasses have to be explicitly identified. Matters are complicated further by the fact that the distinction between operators which only change state in the same problem space, and state elaborations which make temporary changes to the system state is not always obvious. Since the HTA decomposes tasks into operations (and plans) these operations correspond to the Soar

operators (and elaborations). There is some tension here between the level of decomposition and the functionality of operators: a single operator can be used to perform multiple operations.

Another crucial aspect is the determination of the contents of the initial states of the model–there is an initial state for each problem space. Essentially the initial state comprises all the knowledge that the model has at that time about the current state of the world. As well as the initial state for each problem space, there is also a desired (goal) state, which has to be recognised in order to terminate the problem solving episode. For problem spaces below the top level, the initial state may be created by copying the relevant parts of the state from the problem space immediately above the selected one.

4.1 PSCM Constructs for the EW Task Model

The central tenet underlying the Soar architecture is that problem solving is performed by selecting and applying operators to states, within problem spaces. In many cases there may be more than one candidate operator, so some means of choosing between the possible operators is required, since only one operator can apply at any one time. Soar implements this selection process using a preferences mechanism which is described in *The Soar User's Manual* (Congdon & Laird, 1996).

The PSCM level problem spaces, operators, and elaborations that are required in the model are summarised in Table 2 below. These components are organised in terms of problem spaces. The operators and elaborations are explicitly identified, and, where necessary, a short description of their function in terms of how they affect the state of the model is also provided.

Table 2 PSCM Problem Spaces, Operators, and Elaborations for the EW model.

Problem Space	**Operators/Elaborations**	**Description**
Do EW Task	Operator: *Handle Missiles*	Causes space transition to *Do Handle Missiles*.
	Operator: *Track Helicopter*	Causes space transition to *Do Track Helicopter*.
Do Handle Missiles	Operator: *Check for Missiles*	Causes a space transition to *Do Check for Missiles*.
	Operator: *Decoy missiles*	Applies to *Missiles Detected* state, causes a transition to *Decoying Missiles* state.

Do Check for Missiles	Operator: *Check for new missiles*	Adds new missile structures to state, tagged with track number.
	Operator: *Check old missiles*	Removes missiles from the state when current bearing is different from previous one.
Do Decoy Missiles	Operator: *Select missile*	External command.
Do Decoy Missiles (cont'd)	Elaboration: *Get missile details*	Applies to *Decoying Missiles* state. Adds the details of a missile to the state.
	Elaboration: *Get relative wind heading*	Applies to *Decoying Missiles* state. Adds current relative wind heading to the state.
	Operator: *Deploy chaff*	Applies to *Decoying Missiles* state, causes a transition to initial state of *Do Deploy Chaff* space.
	Operator: *Deploy jammer*	Applies to *Decoying Missiles* state, causes a transition to initial state in the *Do Deploy Jammer* space.
	Operator: *Check for chaff bloom*	External command.
Do Deploy Chaff	Operator: *Calculate Escape Route*	Applies to initial state to set up desired values for heading and speed.
	Operator: *Select Chaff Dispenser*	Choose which of the four chaff dispensers to use based on the relative wind and the heading of the missile.
	Operator: *Select Chaff Rounds*	Select the appropriate number of chaff rounds from the display.
	Operator: *Arm Chaff rounds*	Prepares the chaff for firing. Must always follow selection of chaff rounds and precede the firing of the selected rounds.

	Operator: *Fire Chaff rounds*	Causes the appropriate number of chaff to be fired from the selected dispenser. Reduces the level of remaining chaff.
	Operator: *Follow Escape Route*	Applies to initial state to set the desired speed and heading for own ship.
Do DeployChaff (cont'd)	Operator: *Refine escape route*	Establishes a more accurate escape route to follow.
Do Deploy Jammer	Operator: *Select Jammer*	Decide which jammer to use based on the bearing of the missile.
	Operator: *Request permission for jammer*	External command.
	Operator: *Activate Jammer*	Makes the chosen jammer active.
Do Track Helicopter	Operator: *Get helicopter details*	Searches to find the helicopter on the display, and get its bearing and distance.
	Operator: *Control ship heading*	Applies to *Initial* state. Changes own ship heading.
	Operator: *Control ship distance from helicopter*	Applies to *Initial* state. Changes own ship speed.
Do Control Ship Heading	Operator: *Get Helicopter Bearing*	External command.
	Operator: *Read Ship Heading*	External command.
	Operator: *Change Ship Heading*	External command.
Do Control Ship Distance	Operator: *Estimate ship separation*	External command.
	Operator: *Increase ship speed*	External command.
	Operator: *Reduce ship speed*	External command.

As has already been noted, the choice between whether to use an operator, or a state elaboration is a complex one. One way of deciding between them is to consider the resultant changes that will be made to the state. For instance, in Table 2, the action *Get Missile Details* only makes a transitory change to the state: when decoying a missile, the knowledge relating to the details of that missile needs to be available (on the state); once the decoy process has been performed, that information should no longer be available. The difficulty is in deciding how (and when) knowledge should be removed from the state: if it should be removed automatically, when some set of conditions no longer apply, then an elaboration is required; if, on the other hand, it should only be explicitly removed, then an operator should be used.

It could also be argued that changes which involve reading information from the display are more likely to be elaborations, whereas changes involving updating the displayed information are more likely to be operators. This can be justified on the grounds that reading information does not change the state of the world, and is often used as a way of simply gathering information to make a decision, whereas writing information makes a change which can only be undone by some deliberate action. The issue is complicated, however, by the need to interact with the outside world, since this interaction is attained using external operators. The external operators implement the application of task independent knowledge, and as such, are task independent themselves. This means that they can be applied in all problem spaces.

Each of the problem spaces and the associated operators and states for the EW Task are described in more detail below.

4.1.1 *Do EW Task* Problem Space

The *Do EW Task* problem space is the top level problem space for the task. Figure 1 shows the relationships between the states and operators that constitute the *Do EW Task* problem space. The task is basically composed of two parts: trying to handle missiles, and trying to track the helicopter, so there are two possible operators which can be applied:

Handle Missiles. When an attempt is made to apply this operator in the *Do EW Task* problem space, it will lead to a transition to the initial state in the *Do Handle Missiles* problem space.

Track Helicopter. When an attempt is made to apply this operator in the *Do EW Task* problem space it causes a transition to the initial state in the *Do Track Helicopter* problem space.

Since there are two operators that can be applied, some selection criterion is needed to choose between them. In general, it is deemed better to protect own ship and its crew rather than make sure that the helicopter is being closely followed, so the *Handle Missiles* operator should be preferred to the *Track Helicopter* operator whenever a choice has to be made.

4.1.2 *Do Handle Missiles* Problem Space

The *Do Handle Missiles* problem space is where the model deals with hostile missiles. Figure 2 shows the relationship between the states and operators that make up the *Do Handle Missiles* problem space. In order to do this task the model checks to see if there are any missiles, via the *Check for Missiles* operator. If any missiles are detected, a count of the number of missiles is recorded, and then an attempt is made to decoy each missile in turn, using the *Decoy Missiles* operator. The missiles can be decoyed in a sequential manner because there is only one type of missile in the EW Task scenarios, so no single type of missile is more dangerous than any other type.

Check for Missiles. This operator is applied to the initial state in the problem space. It involves searching the display to see whether there are any hostile missiles present, and counts them all. The task is somewhat simplified by the fact that all missiles are of the same type, and travel at the same relative speed. Each missile can be uniquely identified by its track number, so a simple way to check whether a missile has previously been seen is to test whether its track number is already known.

Decoy Missiles. This operator is applied if there are any missiles which have not yet been

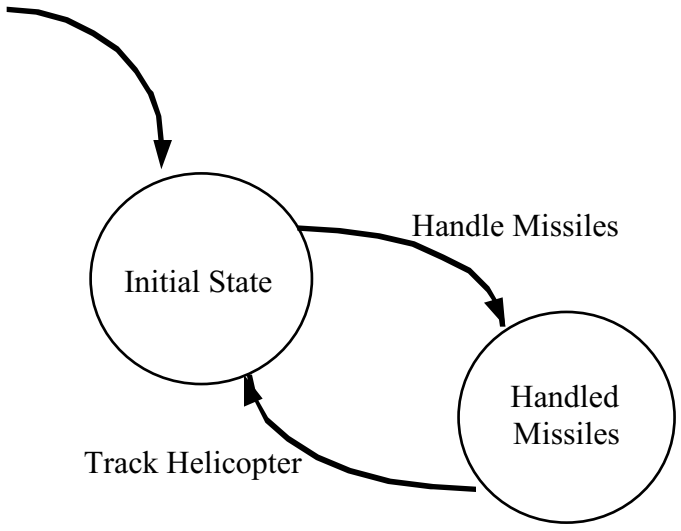

Figure 1. Do EW Task Problem Space: States and Operators that perform transitions between states.

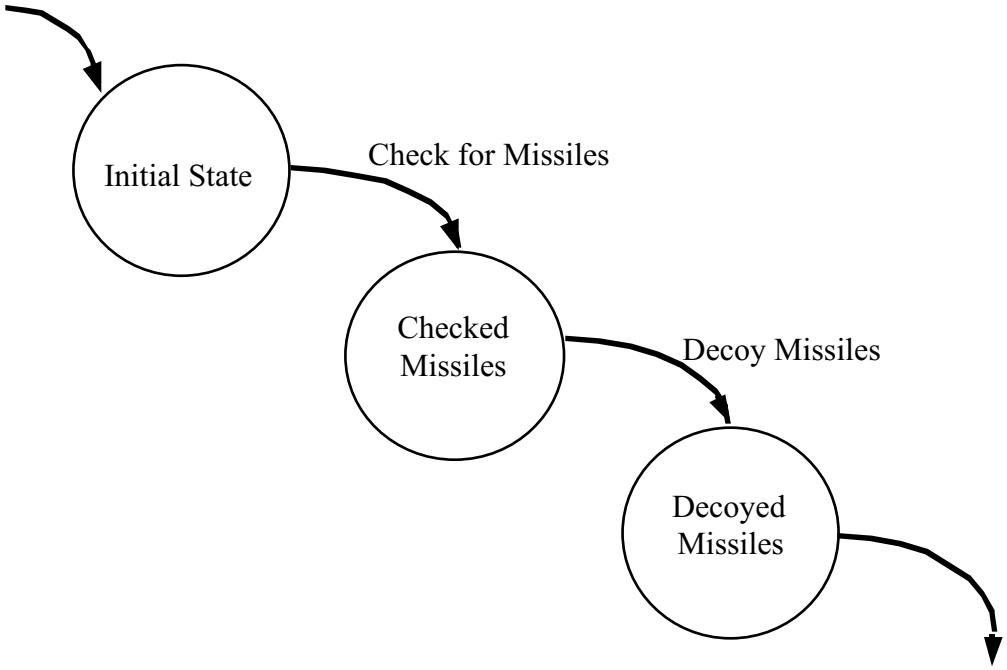

Figure 2. Do Handle Missiles Problem Space: States and Operators that perform transitions between states.

decoyed. It causes a change to the initial state of the *Do Decoy Missiles* problem space. In general missiles can only be decoyed one at a time, so this operator is applied once per missile.

4.1.3 *Do Check for Missiles* Problem Space

This problem space is used to implement the process of searching for missiles. The relationship between the operators and states that make up the *Do Check For Missiles* problem space are shown in Figure 3. There are essentially two parts to the sub-task of checking for missiles. The first involves looking for new missiles which have appeared since the last time the screen was scanned. The second involves checking the old missiles to see which ones have been successfully decoyed. Once a missile has been successfully decoyed, it is ignored. The operators used to achieve the process of checking for missiles are:

Check for New Missiles. This involves the use of external commands. For each missile detected, a new substructure is added to the state, which includes the track number for that missile. The track number is used to uniquely identify each missile.

Check Old Missiles. This involves the use of external commands. The tracks of existing missiles are checked, to see whether there has been a noticeable deflection in the bearing of the missile. A deflection indicates that the missile is now tracking the chaff, rather than own ship, so the missile can be ignored in future checks because it is regarded as having been successfully decoyed.

4.1.4 *Do Decoy Missiles* Problem Space

The *Do Decoy Missiles* problem space is where the task to decoy individual missiles is carried out. The relationships between the operators and states that make up the *Do Decoy Missiles* problem space are shown in Figure 4. Before a missile can be decoyed the details of that missile are required, along with the relative wind information. Where possible, all available countermeasures are used. This means that chaff is always used, and permission to use the appropriate jammer is requested unless that jammer is already in use. In order to decoy a missile using the countermeasures, it must first be selected using the mouse. Once a missile has been selected, the *Deploy Chaff* and *Deploy Jammer* operators can be proposed and applied. The *Deploy Chaff* and *Deploy Jammer* operators are always applied.

Select Missile. This is an external command which has to be used to select a missile to be decoyed. This involves clicking the mouse button on the missile, which causes a white ring to encircle the missile icon on the display.

Get Missile Details. This is a state elaboration which reads the missile details from the display screen. Since this involves perceiving the contents of the display screen, the eye extension to the model is used, which is implemented using external commands. The eye extension provides a description of the missile that is comparable to the level of detail normally found in the track table on the two head version of the OOPSDG simulation. So, for each missile there will be an associated description that appears on the state. The only information which is directly available from the display is the track number.

Get Wind Details. This is a state elaboration which reads the relative wind value from the display. As with the *Get Missile Details* elaboration, this elaboration is implemented using external commands.

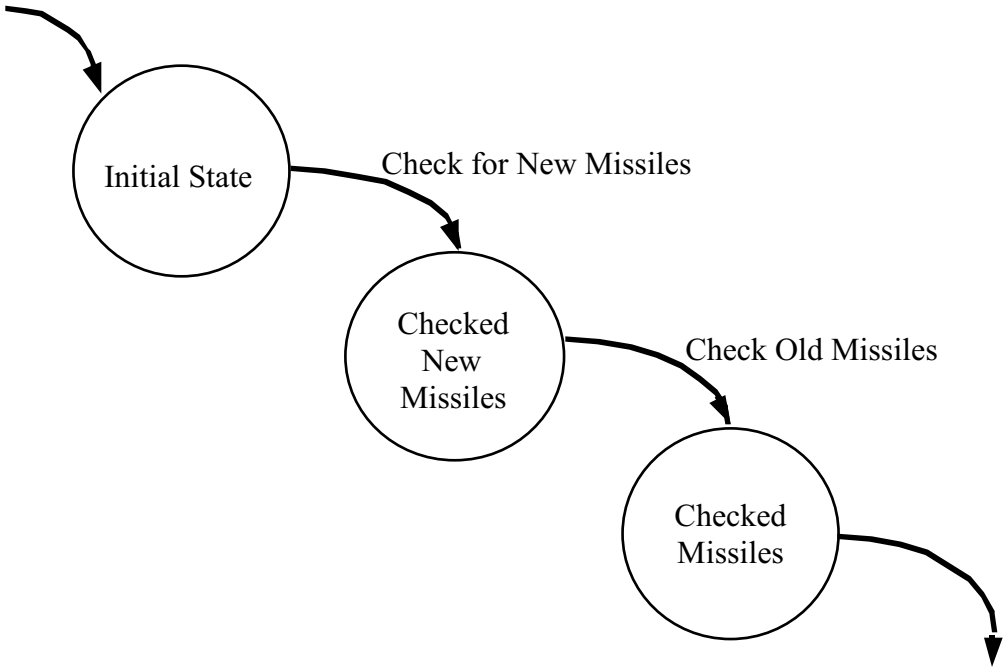

Figure 3. Do Check For Missiles Problem Space: States and Operators that perform transitions between states.

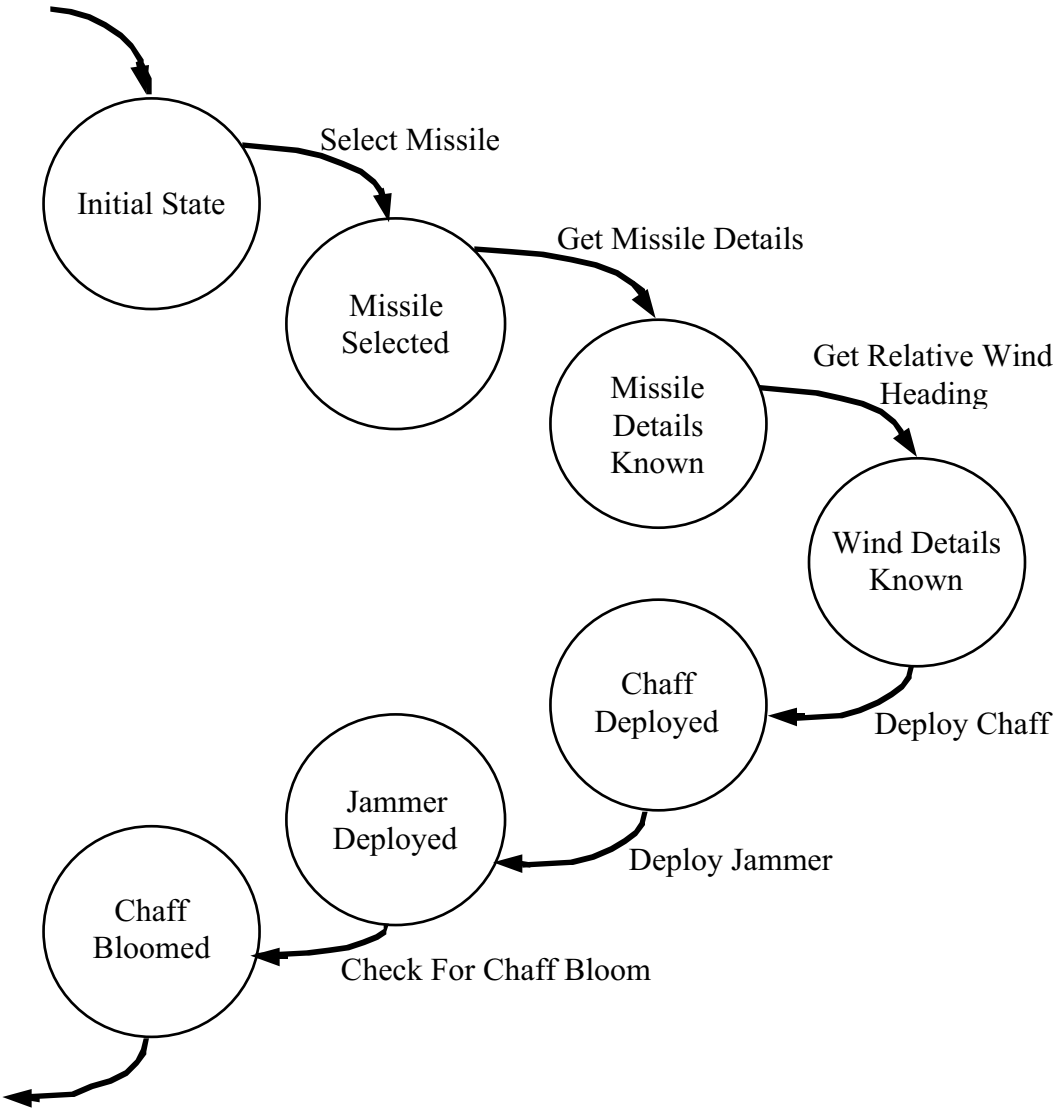

Figure 4. Do Decoy Missiles Problem Space: States and Operators that perform transitions between states.

Deploy Chaff. This operator causes a transition to the *Do Deploy Chaff* problem space.

Deploy Jammer. This operator causes a transition to the *Do Deploy Jammer* problem space.

Check for Chaff Bloom. This is an external command, which involves looking at the display in order to make sure that a white filled circle appears where the chaff was deployed. The presence of this circle indicates that the chaff has successfully bloomed, a process which takes about 15 seconds.

4.1.5 *Do Deploy Chaff* Problem Space

This is the problem space used to implement the task of decoying an individual missile using chaff. The relationships between the operators and states that make up the *Do Deploy Chaff*

problem space are shown in Figure 5. The first step in this task is to select a missile. Once this has been done, an approximate escape route is calculated, then the chaff is deployed (selected, armed and fired). The planned escape route is then followed, and a check is carried out to make sure that the chaff bloomed successfully. If the chaff fails to bloom, another round of chaff is launched. Once the chaff has been successfully deployed, the ship's course is refined to a more optimal one.

The individual operators in the *Do Deploy Chaff* problem space are:

Calculate Escape Route. This operator calculates the approximate speed and heading required to evade a particular missile. Normally the speed will be set as high as possible, but the heading will depend on several factors. The initial escape route is based on a more qualitative assessment of the situation. The aim is to initially try to buy some time, and worry about refining the course and speed of own ship once countermeasures have been deployed.

Select Chaff Dispenser. This operator is used to choose which chaff dispenser is most appropriate for the missile which is to be decoyed. The selection will be made based on the bearing of the missile, the direction in which the ship is facing (since the port side is *always* on the left of the ship and the starboard side is *always* in the right), and the relative wind. This is really a mental operator, since it effectively determines which set of chaff rounds should be selected, rather than carrying out any observable action.

Select Chaff Rounds. The physical selection of the chaff rounds is implemented using the hand extension to the model, which involves the use of external commands.

Arm Chaff Rounds. This operator causes the selected chaff rounds to be loaded ready for firing. The status of the selected chaff dispenser changes to armed. The operator is implemented using external commands, since it involves clicking on the command button labelled "Arm".

Fire Chaff Rounds. This operator causes the chaff to be fired, and changes the state of the dispenser from armed to fired. *Fire Chaff Rounds* is implemented using external commands, since it involves clicking on the command button labelled "Fire".

Follow Escape Route. This operator implements the initial calculated escape route by setting the new course and new speed for own ship. This is achieved using external commands which drive the hand extension to the model. The new course and new speed are set to the worked out by the *Calculate Escape Route* operator.

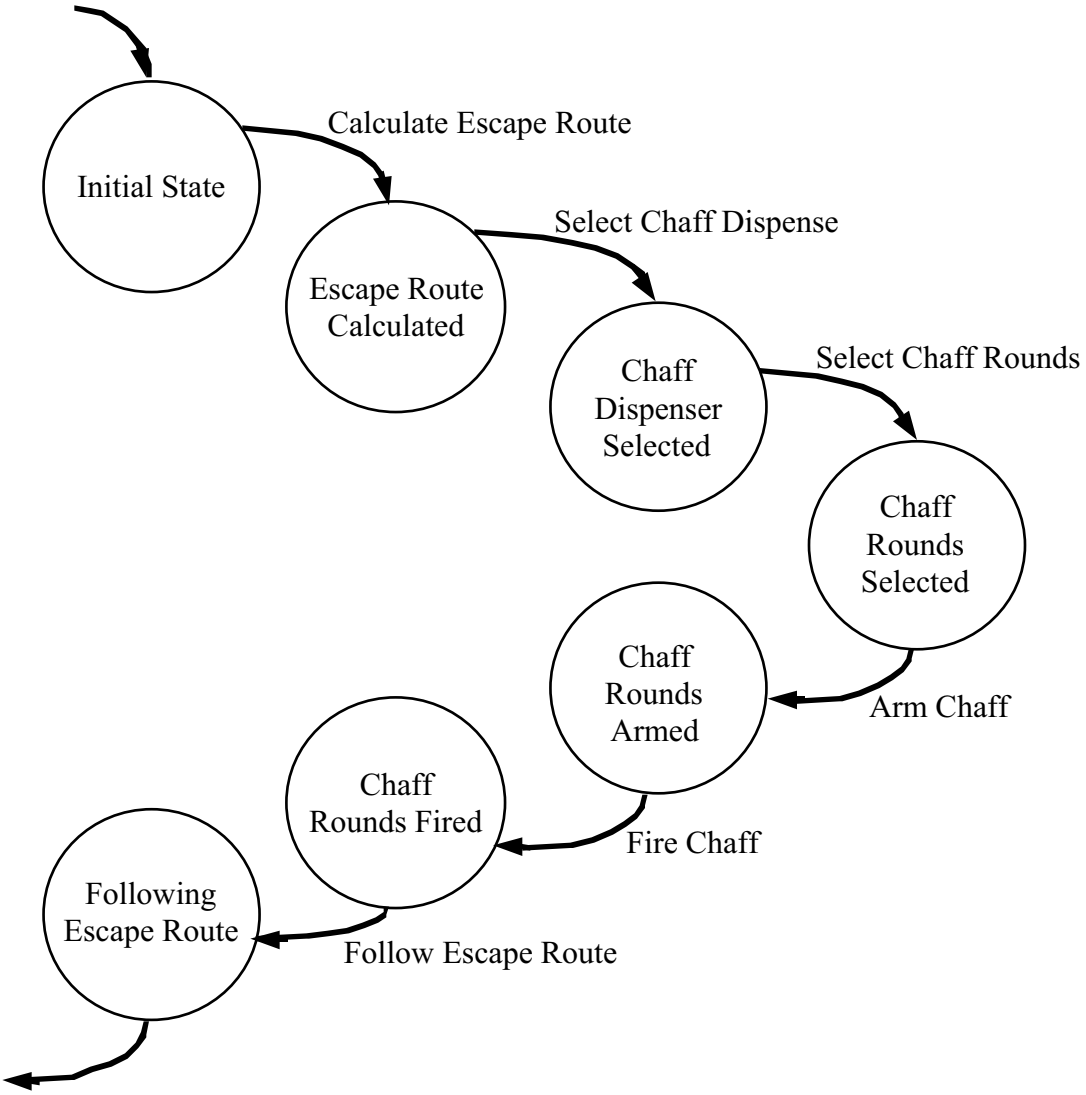

Figure 5. Do Deploy Chaff Problem Space: States and Operators that perform transitions between states.

Refine Escape Route. This operator is implemented using external commands, and involves updating own ship heading to a more optimal course. The new value is calculated using the missile heading, and the relative wind. Normally, no change in ship speed will be required.

4.1.6 *Do Deploy Jammer* Problem Space

This is the problem space that implements the task of decoying an individual missile using one of the on-board jammers. The relationship between the operators and states that make up the *Do Deploy Jammer* problem space are shown in Figure 6. Deploying the jammer is a comparatively straightforward task. The first step is to decide which jammer is on the same side of own ship as the missile. The jammer is checked to see if it is already active. If it is not active, permission must be sought before it can be activated, since activating a jammer makes own ship more detectable by radar. Once permission has been granted, it is simply a

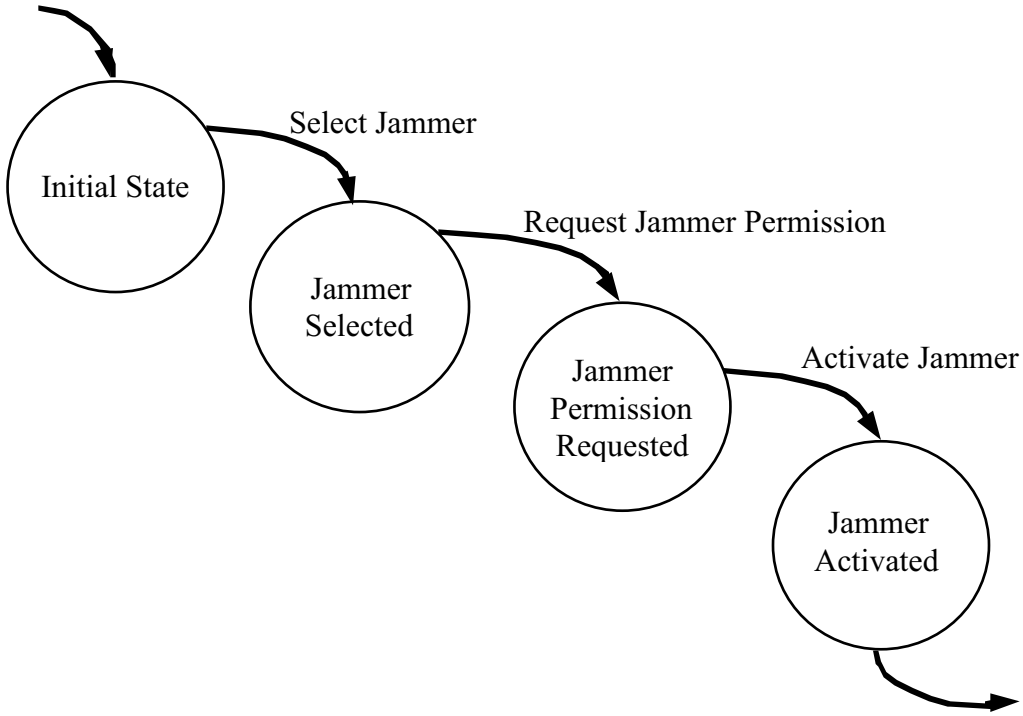

Figure 6. Do Deploy Jammer Problem Space: States and Operators that perform transitions between states.

---

matter of selecting the jammer on the same side of own ship as the missile, and then activating the jammer. If the selected jammer is already active, or permission is not granted, the task is terminated.

The individual operators required in the *Do Deploy Jammer* problem space are:

Select Jammer. The decision is made to select the port side jammer or the starboard side jammer, based on the bearing of the missile and the current heading of the ship. This is effectively a mental operator.

Request Jammer Permission. This operator involves asking for permission to use the jammer. For the EW task, the granting of permission is dependent upon the scenario, and the missile involved. The pre-specified scenarios for the task are set up in such a way that the jammer can only be used for certain missiles.

Activate Jammer. This involves a sequence of operations, implemented using external commands. The track number for the missile has to be typed into the appropriate box on the display. The status of the selected jammer is changed from off to on.

4.1.7 *Do Track Helicopter* Problem Space

The *Do Track Helicopter* problem space is used to implement the task of maintaining station with the helicopter. The relationship between the operators and states for the *Do Track Helicopter* problem space are shown in Figure 7. Tracking the helicopter involves updating

own ship's heading and maintaining the separation between own ship and the helicopter within predefined limits. The operators used to perform this task are:

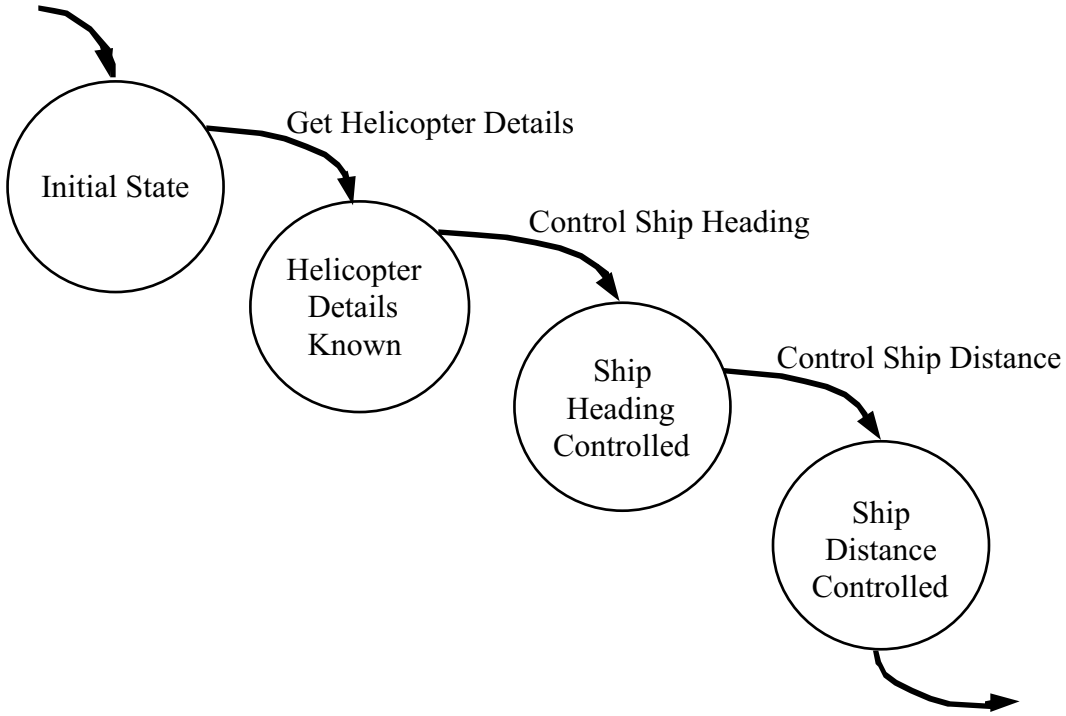

Figure 7. Do Track Helicopter Problem Space: States and Operators that perform transitions between states.

Get Helicopter Details. This is implemented using external commands, and involves searching the display to find the helicopter.

Control Ship Heading. This causes a transition to the *Do Control Ship Heading* problem space.

Control Ship Distance. This causes a transition to the *Do Control Ship Distance* problem space.

4.1.8 *Do Control Ship Heading* Problem Space

This problem space is used to implement the task of updating the heading of own ship, based on the current bearing of the helicopter from own ship. The aim is to keep the helicopter at a bearing of 90° from the ship. The relationships between the operators and states that make up the *Do Control Ship Heading* problem space are shown in Figure 8, which shows that the task of controlling the ship's heading is achieved using three operators.

Get Helicopter Bearing. This operator is implemented using an external command, and simply adds the bearing of the helicopter to the state.

Read Ship Heading. This operator, which is implemented using external commands, simply reads the current course of the ship from the screen and adds it to the state.

Change Ship Heading. The ship heading is set to the bearing of the helicopter. The new course attribute on the state is set to the new value. This involves making use of the hand extension to the model, and is implemented using external commands.

4.1.9 *Do Control Ship Distance* Problem Space

This problem space implements the task of keeping own ship at a distance of 2.5 miles from the helicopter. The relationships between the operators and states that make up the *Do Control Ship Distance* problem space are shown in Figure 9. The task of controlling the distance between own ship and the helicopter is achieved using three operators.

Estimate Ship Separation. This operator is used to determine the distance between own ship and the helicopter. It is implemented using external commands.

Increase Ship Speed. The ship speed is set according to the size of the distance between own ship and the helicopter. If the distance exceeds 2.5 miles (plus some tolerance value) then the speed is increased This operator is implemented using external commands. (Note that in reality the expert achieved the goal of this task by circling round so that the helicopter could

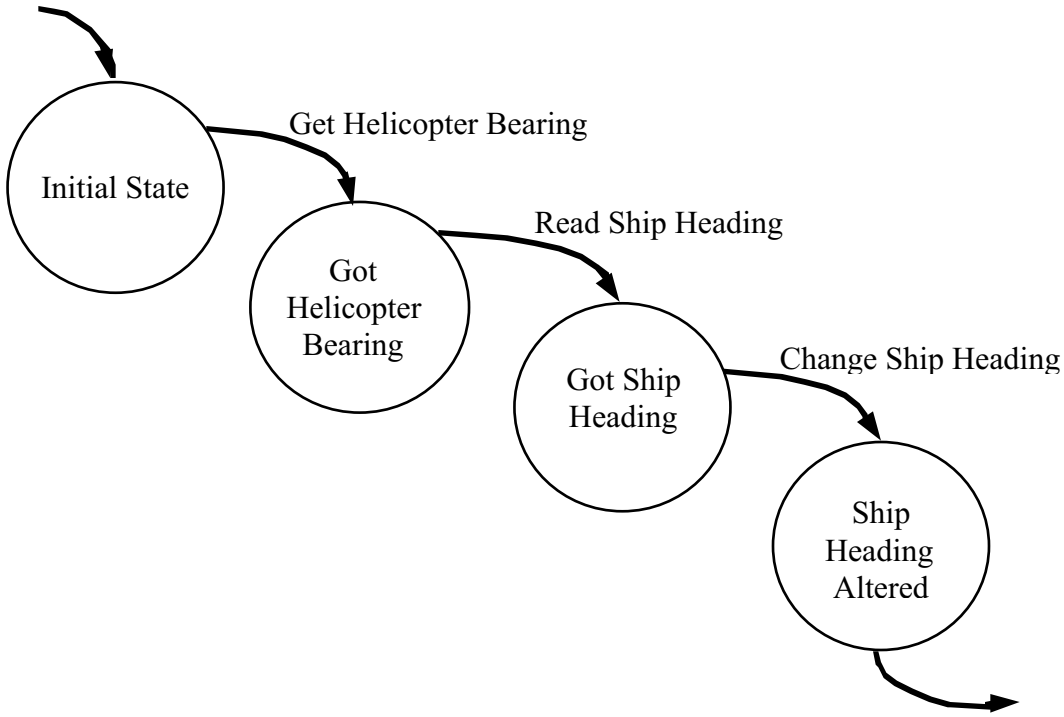

Figure 8. Do Control Ship Heading Problem Space: States and Operators that perform transitions between states.

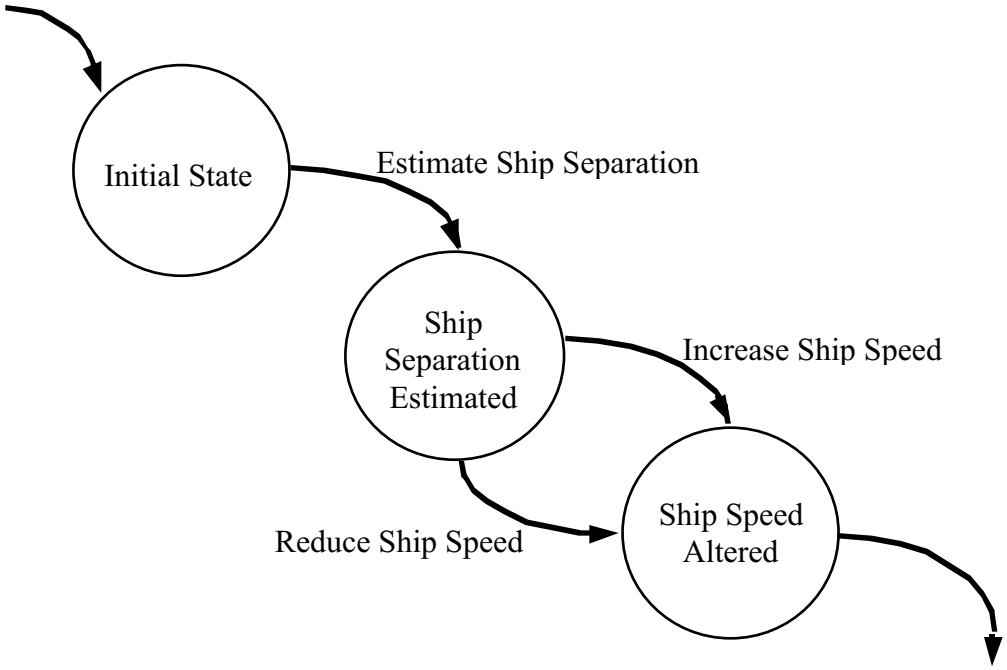

Figure 9. Do Control Ship Distance Problem Space: States and Operators that perform transitions between states.

---

catch up with it.)

Reduce Ship Speed. If the distance between own ship and the helicopter is less than 2.5 miles (less some tolerance value) then the speed of own ship is correspondingly reduced. This operator is implemented using external commands.

4.2 Initial States and Knowledge

The contents of the initial states for each of the problem spaces can be identified using the information from Table 2 shown above, constrained by the requirements imposed by the Knowledge Level model as represented by the elaborated HTA shown in Table 1. The initial state represents the base level of knowledge for a particular problem space, so it contains values for parameters that are either fixed or have default values for the current instance of the task associated with that problem space.

4.2.1 *Do EW Task* Initial State

It is assumed that at the start of the task, the model is aware of all of the values displayed on the screen.

- Ship speed
- Ship heading
- Relative wind
- True Wind
- Chaff in use

- Jammer in use
- Missile detected
- Helicopter distance
- Helicopter bearing

Knowledge is also present in the initial state which allows the model to decide which sub-task should be preferred in particular circumstances. The model prefers to do the *Handle Missiles* sub-task, when it is in the initial state, and prefers the *Track Helicopter* task when it is in the handled missiles state. The sub-tasks are implemented as Soar operators.

4.2.2 *Do Handle Missiles* Initial State

The *Handle Missiles* sub-task involves some exploratory search to gather information which is used as the basis for carrying out the sub-task. The initial state for *Do Handle Missiles* does not contain any task knowledge. All of the knowledge required to do the sub-task is gathered each time the sub-task is performed.

4.2.3 *Do Check for Missiles* Initial State

The *Check for Missiles* sub-task involves a visual search of the display in order to determine the presence or otherwise of missiles. The initial state of *Do Check for Missiles* therefore does not contain any task knowledge.

4.2.4 *Do Decoy Missiles* Initial State

The *Decoy Missiles* sub-task involves selecting a missile, collating the wind and missile information, and then deploying the appropriate countermeasures. All of the task-specific information needed to do the task is available from the display screen. The initial state of *Do Decoy Missiles* therefore does not contain any task knowledge.

4.2.5 *Do Deploy Chaff* Initial State

The *Deploy Chaff* sub-task involves selecting, arming, and firing chaff against a particular missile, as well as calculating and following an escape route that maximises the separation between the chaff cloud and own ship. The initial version of the model has escape route information hard-coded into it, so there is no knowledge on the initial state for *Do Deploy Chaff*. In later versions, this is liable to change, with missile details being added to the initial state, so that the escape route can be calculated.

4.2.6 *Do Deploy Jammer* Initial State

The *Deploy Jammer* sub-task involves requesting permission to use the jammer, and, if this is granted, selecting and activating the relevant jammer. Although the sub-task involves use of missile details, these are retrieved directly from the top state. There is consequently no task knowledge on the initial state for *Do Deploy Jammer*.

4.2.7 *Do Track Helicopter* Initial State

The *Track Helicopter* sub-task involves reading information about the helicopter from the

helicopter details have to be read from the display each time the task is performed. There is consequently no task knowledge on the initial state for *Do Track Helicopter*.

4.2.8 *Do Control Ship Heading* Initial State

4.2.9 *Do Control Ship Distance* Initial State

Part 4: The Soar Implementation Model aka The Symbol Level Computational Model (SLCM)

5. Developing the Soar Model

Soar PSCM models are implemented as production rule models using the Soar language, the resultant model is sometimes referred to as the Symbol Level Computational Model (SLCM). One of the inherent drawbacks of any production rule system that allows rules to fire in parallel is that it is difficult to statically determine the flow of execution of the model, since rules can fire dynamically based on the current status of the model. There are, however, a number of techniques that can be used to help to make it easier to understand models, and manage their development in a coherent manner. These are described below.

5.1 Production Rules

The Soar model implements the PSCM level model in an executable form, in this case, Soar production rules. These productions take two general forms. The most general one is the standard production rule (or condition–action) format, where the situation is represented as a set of conditions on the left hand side of the rule. When these conditions obtain, the actions specified on the right hand side are performed:

if (situation) then (action)

The other format involves extending the state by adding objects to it (state elaborations in Soar terminology). When the conditions on the left hand side of the rule are satisfied, the state is elaborated with the objects that appear on the right hand side of the rule:

if (situation) then (situation)

The distinction between the two is somewhat blurred in Soar, but generally productions take the second form. The only exceptions are operator application rules, and rules which explicitly require operator-support, in order to make sure that the changes effected by the rule are permanent.

5.2 Naming Soar Productions

In order to make it easier to follow the behaviour of the model as it runs, it is useful to adopt a naming convention for all productions. The following template provides one example of the sort of structure to use:

*ProblemSpaceName*StateName*OperatorName*Action*

This makes it easier to find the source code file for a particular production, for example, since it can be determined directly from the *ProblemSpaceName* part of the name of the rule. The main drawback is that the names of productions can easily become unwieldy, running to 50 or more characters.

5.3 Naming Soar States

In order to facilitate comprehension of the Soar model, the state in each production can be tagged with a name attribute. Since the only way to make permanent changes to the state in Soar is by using operators, the application of an operator is the only way that the name of a state can be modified. In this way, it becomes possible to track the behaviour of the Soar model as a kind of finite state machine. There are subtle limitations, however, in that using an operator to change the name of a state does not actually generate a new state, it simply modifies the name of the old state. Since Soar states are really hybrid state/problem space constructs, generating a new state involves generating another level in the hierarchy of state/problem space constructs.

5.4 Organising the Soar Source Code Files

The EW Task model is constructed by implementing each problem space in a separate file. There are a number of reasons for doing this:

- It means that the structure of the model is mirrored by the structure of the code.
- It makes the problem spaces self-contained.
- It makes it easier to investigate the effects of not loading a particular problem space.

A case could be made for more closely mirroring the model structure by having one directory per problem space, and one file per operator. This structure may get too cumbersome for projects with many problem spaces, and requires the deliberate use of the problem space construct. Within the more recent versions of Soar, the use of problem spaces can be implicit, in that the problem spaces are not separately identified. On smaller projects, however, it may be useful to map the task structure onto separate directories and files, since the hierarchical directory structure of the project would directly mirror the hierarchical structure of the model.

6. Implementation Considerations

There are a number of aspects that need to be considered when implementing Soar models. Many of these constrain the implementation to be done in a particular way. The particular aspects that were considered during the implementation of the EW task are described below. The majority of these considerations are related to the need for the model to interact with an external simulation. By default, Soar provides little support for so-called *external interaction*.

Task performance is constrained by the need to interact with the task's application interface. Primarily this involves scanning the display, selecting objects on the display using the mouse, and entering values using the keyboard. In order to accomplish this level of interaction, the cognitive model will need to be able to utilise mechanisms that provide the model with visual perception, and motor action (Baxter & Ritter, 1996a; 1996b). These mechanisms are being implemented in SL-GMS and will be integrated with the model when they become available. The initial version of the model will have to be implemented on the assumption that the mechanisms are to be made available at some stage in the future.

The OOPSDG simulation and interface provide much more functionality than that used in the experimental task. The extra functionality of the simulation will not be encapsulated in the model. The extra functionality of the interface, however, will need to be handled in some way, because the model will be interacting with the display. The model will have to be able to take account of objects on the display that it does not need to use to perform the task. There is a part of the display that provides information on the ship's radars, for example, which is not required to do the task. The simplest solution is for the model to effectively ignore such details.

The model will typically outperform users. There are two main reasons behind this. The first is that Soar always brings to bear all the knowledge it can muster in order to solve a particular problem. In contrast, people can *have* the appropriate knowledge, but they do *not* always apply it, this is generally referred to as the problem of *inert knowledge*. The second reason is that Soar models do not generally make the sorts of erroneous actions that are inherent in human performance. Even experts perform erroneous actions, although they usually detect them and recover from them before any undesirable consequences arise.

The difficulty of choosing between whether to use operators or state elaborations has already been discussed. The problem becomes even more significant when the Soar model has to communicate with the external environment. If the EW Task simulation was simply written in Soar productions, then the reading in of data values could quite validly be achieved using elaborations. Since the EW Task is external to the Soar model, however, and the only way to communicate with the external environment is using the eye and hand extensions. The model has to explicitly control these

extensions, and since this involves moving the hand and eye to new positions, as appropriate, they are implemented as operators. Thus anything that appears at the PSCM level as an elaboration has to be implemented using operators whenever it involves having to interact with the external world. If it is not, then it may lead to strange behaviour in the model, such as the eye moving back to its original position when the initiating conditions for a saccadic movement of the eye no longer apply.

External I/O within Soar is accomplished using the IO substructure of the top state of the model. The IO substructure has an input link and an output link. Data destined for the outside world is explicitly placed onto the output link substructure. Data arriving from the outside world appears on the input link substructure. The Soar model has to manage the use of the input link and output link, copying data as appropriate, and removing old data. The details on how to use the Soar I/O mechanism are fully described in *The Soar Advanced Applications Manual* (Congdon and Schwamb, 1997).

Communication with the hand and eye extensions is implemented using a general purpose Unix socket utility (Ong & Ritter, 1994). This utility allows data to be passed between the model and the hand and eye extensions as lists of attribute-value pairs.

The fact that Soar is now a mode within Tcl/Tk makes it possible to call Tcl/Tk procedures from within Soar. The integration of Soar with external models is a time consuming and often messy procedure. As a first pass, it is possible to make the model behave in a manner somewhat similar to HOS (Wherry, 1976). This means that rather than physically carry out interactions, the model simply calls a Tcl/Tk procedure which implements a delay of the appropriate length, before instantaneously doing the action. Timings for these delays can be based on existing data from the Keystroke Level Model (Card, Moran & Newell, 1980), for example, as a first approximation.

7. The Soar Productions

Each of the Soar productions is described below, in terms of the conditions and actions that comprise the rule. These are spelt out in general terms, rather than using the Soar language, although the conjunctions of the condition part of the rules are emphasised by the use of upper case text for the word “and”.

In Soar (up to, and including version 7, the current version) operators are implemented in three stages. First they are proposed, then they are applied, and finally they are terminated. So for each operator, there is a set of productions, which can be divided into three subsets, one for each of the stages:

- *Operator Proposal:* An operator must be proposed before it can be applied. It can be proposed under different conditions, so there may be several operator proposal productions.

- *Operator Application:* An operator can only applied if it goes into the operator slot on the state, in other words if it is selected as being the best operator out of all of those that are currently proposed. Normally there will only be one operator application production for each operator.

- *Operator Termination:* When an operator has been applied, it has to be explicitly terminated. An operator termination production is used to do this, and normally there will only be one of these per operator. In general the operator termination production tests the value of an attribute on the state which has been set by the operator, and then changes the preferences associated with the operator to a reconsider preference

In general, the conditions for the application of an operator are a subset of those for the proposal of that operator, combined with the need for the operator to be in slot on the state. The subset may be the full set of conditions for the operator proposal, and is usually a non-empty subset.

7.1 Do EW Task Problem Space

There is a tension between the *Handle Missiles* operator and the *Track Helicopter* operator in that they are both important tasks but the decision on which one to choose is not well-defined. In those instances where it is possible for either of the operators to be applied, the *Handle Missiles* operator is preferred to the *Track Helicopter* operator.

7.1.1 Handle Missiles Operator

Proposed: When the model is in the *Do EW Task* problem space. The operator is added to the state with an acceptable preference attached to it.

Applied: When the model is in the *Do EW Task* problem space AND the operator is in slot, an operator no-change impasse occurs. Application of this operator leads to a transition to *Initial State* of the *Do Handle Missiles* problem space. The *Do Handle Missiles* problem space implements the *Handle Missiles* operator.

Terminated: When a decoy plan has been successfully implemented for a missile, such that the model is in the *Handled Missiles* state in the *Do EW Task* problem space.

7.1.2 Track Helicopter Operator

The proposal conditions for this operator require the target values to be within a given range. So for the tolerance on the difference between own ship's heading and the bearing of the helicopter from own ship, it may be ±2°, and for the distance it may be ±0.5 miles. The tolerances on the values reflect the inherent difficulty of maintaining an exact heading and separation when the prevailing conditions (sea and wind) continually affect the ship and the helicopter motion.

Proposed: When there is a difference between own ship heading and the bearing of the helicopter AND the approximate distance between own ship and the helicopter is more than 2.5 miles AND there are no missiles.

Applied: When the model is in the *Do EW Task* problem space AND the operator is in slot, an operator no-change impasse occurs. Application of this operator leads to a transition to *Initial State* of the *Do Track Helicopter* problem space. The *Do Track Helicopter* problem space implements the *Track Helicopter* operator.

Terminated: When a plan has been implemented for regaining station with the helicopter, and the model is back in the *Initial State* of the *Do EW Task* problem space.

7.1.3 Operator Selection

The *Track Helicopter* operator is preferred to the *Handle Missiles* operator when the model is in the *Handled Missiles* state of the *Do EW Task* problem space. This occurs when all of the missiles have been successfully decoyed.

The *Handle Missiles* operator is preferred to the *Track Helicopter* operator in all other instances. In other words, *Handle Missiles* takes precedence when the model is in the *Initial State* of the *Do EW Task* problem space.

7.2 Do Handle Missiles Problem Space

The EW Officer's Task divides into two basic parts: decoying missiles, and tracking the helicopter. The *Do Handle Missiles* problem space implements the task of decoying of missiles. Missiles are

normally decoyed individually, by deciding which missile to decoy, and then deploying the appropriate countermeasures.

7.2.1 Check for Missiles Operator

The *Check for Missiles* operator causes the main PPI to be scanned in order to determine the presence of missiles. Thus, the external operators which implement the perceptual mechanism for the model are used in order to get the current details for each of the missiles on the display. In addition, the external commands which implement motor actions may be required, depending on the current zoom level that the PPI is showing. The lack of missiles on the main PPI may simple mean that the PPI is currently set at a zoom level relatively close to the ship. In other words, there are missiles in the area, but they are currently located beyond the range currently shown on the display, The only guarantee that there are no (more) missiles is when the zoom level is set to the maximum range (100).

Proposed: When the model is in the *Initial State* of the *Do Handle Missiles* problem space.

Applied: When the model is in the *Initial State* of the *Do Handle Missiles* problem space AND the operator is in slot on the state, an operator no-change impasse occurs. Application of this operator leads to a transition to *Initial State* of the *Do Check For Missiles* problem space. The *Do Check For Missiles* problem space implements the *Check For Missiles* operator.

Terminated: When all the missiles on the display have been checked such that the model is in the *Checked Missiles* state of the *Do Handle Missiles* problem space. This is a difficult condition to detect since it depends on the number of missiles as well as the identifiers of those missiles. The number of missiles on its own is insufficient because the track for one missile may disappear only to be replaced by another, new, missile.

7.2.2 Decoy Missiles Operator

Once the tracks for all of the missiles have checked, the next step is to decoy them. All of the new missiles need to be decoyed. It may also be necessary to send out more countermeasures to decoy missiles which do not appear to have been successfully decoyed.

Proposed: When in the *Checked* Missiles state of the *Do Handle Missiles* problem space.

Applied: When the model is in the *Checked Missiles* state of the *Do Handle Missiles* problem space AND the operator is in slot, an operator no-change impasse occurs. Application of the *Decoy Missiles* operator leads to a transition to *Initial State* of the *Do Decoy Missiles* problem space. The *Do Decoy Missiles* problem space implements the *Decoy Missiles* operator.

Terminated: When the plan for decoying a missile has been carried out, such that the model is in the *Decoyed Missiles* state of the *Do Handle Missiles* problem space.

7.3 Do Check For Missiles Problem Space

The task of checking for missiles essentially has two parts to it. The first involves checking to see whether there are any new missiles that have to be decoyed. The second is to check the known missiles that have not yet been successfully decoyed, to monitor their progress, and look for a deviation in their heading, which indicates that they have been decoyed.

7.3.1 Check For New Missiles Operator

Proposed: When the model is in the *Initial State* of the *Do Check For Missiles* problem space.

Applied: When the model is in the *Initial State* of the *Do Check For Missiles* problem space AND the operator is in slot on the state. Application of the operator leads to a transition from the *Initial State*

to the *Checked New Missiles* state of the *Do Check For Missiles* problem space. It also adds a new missile substructure to the state for each new missile that is detected. This substructure consists of the track number for the missile. The initial version of the model is simplified in that it only ever has to deal with one missile. In later versions, this operator will need to be made more complex to ensure that the details of all of the new missiles are noted.

Terminated: When the check for new missiles is completed, such that the model is in the *Checked New Missiles* state of the *Do Check For Missiles* problem space.

7.3.2 Check Old Missiles Operator

Proposed: When the model is in the *Checked New Missiles* state of the *Do Check For Missiles* problem space.

Applied: When the model is in the *Checked New Missiles* state of the *Do Check For Missiles* problem space AND the operator is in slot on the state AND there is some substructure for the new missiles on the state. The application of this operator leads to a transition from the *Checked New Missiles* state to the *Checked Missiles* state of the *Do Check For Missiles* problem space. This operator also adds some substructure to the superstate for the missile, describing the missile's track number, and its distance and bearing with respect to own ship.

Terminated: When the checking of the old missiles is complete, such that the model is in the *Checked Missiles* state of the *Do Check For Missiles* problem space.

7.4 Do Decoy Missiles Problem Space

The *Do Decoy Missiles* problem space is used to implement the task of deploying countermeasures to try and decoy incoming missiles away from own ship. The task is achieved by selecting a missile, and then launching chaff rounds, and deploying one of the onboard jamming devices against it. The basic state transition diagram showing the relationship between the operators and the states is shown in Figure 4.

7.4.1 Select Missile Operator

Proposed: When the model is in the *Initial State* of the *Do Decoy Missiles* problem space.

Applied: When the model is in the *Initial State* of the *Do Decoy Missiles* problem space AND the operator is in slot on the state. Application of this operator leads to a transition from the *Initial State* to the *Missile Selected* state. The selection of a particular missile is achieved using external operators to move the mouse pointer to a missile and click on it. This causes the icon for selected missile to be surrounded by a white ring on the PPI display. The initial version of the model only has a single missile to deal with, so the *Select Missile* operator is correspondingly simple. In later versions, the operator will have to be expanded to incorporate a method for deciding which missile to select.

Terminated: When a missile has been selected, such that the model is in the *Missile Selected* state.

7.4.2 Get Missile Details Elaboration

This elaboration fires to get the current details for a missile. It is used to get the latest details on the missile for use in working out how to deal with an incoming missile. In order to do this it utilises the external commands to move the model's eye around the PPI to assess the bearing of the missile. Note that in this particular task the bearing of the missile is not directly available, and has to be estimated from the PPI using the missile track.

This is actually an operator. It is an artefact of using a finite state machine approach. You need to keep track of the state changes, which means that they have to be permanent, hence they have to be implemented as operators.

7.4.3 Get Relative Wind Heading Elaboration

This elaboration fires to get the current value for the relative wind heading. It is used in deciding what the new heading of the ship should be. In order to read the heading from the display, the external commands which implement the model's perceptual capability have to be utilised. The model eye has to be moved to the location where the relative wind information is shown on the display. The relative wind information is displayed numerically in the top left hand corner of the PPI, and is also shown as a directional arrow at the centre of the PPI.

This is actually an operator

7.4.4 Deploy Chaff Operator

Proposed: When in the *Wind Details Known* state of the *Do Decoy Missiles* problem space.

Applied: When the model is in the *Wind Details Known* state of the *Do Decoy Missiles* problem space AND the operator is in slot on the state, an operator no-change impasse occurs. Application of the *Deploy Chaff* operator leads to a transition to *Initial State* of the *Do Deploy Chaff* problem space. The *Do Deploy Chaff* problem space implements the *Deploy Chaff* operator.

Terminated: When the chaff has been deployed, such that the model is in the *Chaff Deployed* state.

7.4.5 Deploy Jammer Operator

Proposed: When the model is in the *Chaff Deployed* state of the *Do Decoy Missiles* problem space.

Applied: When the model is in the *Chaff Deployed* state of the *Do Decoy Missiles* problem space AND the operator is in slot on the state, an operator no-change impasse occurs. Application of the *Deploy Jammer* operator leads to a transition to the *Initial State* of the *Do Deploy Jammer* problem space. The *Do Deploy Jammer* problem space implements the *Deploy Jammer* operator.

Terminated: When the jammer has been activated, such that the model is in the *Jammer Deployed* state of the *Do Decoy Missiles* problem space.

7.4.6 Check For Chaff Bloom Operator

Proposed: When the model is in the *Jammer Deployed* state of the *Do Decoy Missiles* problem space.

Applied: When the model is in the *Jammer Deployed* state of the *Do Decoy Missiles* problem space AND the operator is in slot on the state AND the model is still in the *Initial State* of the *Do EW Task* problem space. Application of the *Check For Chaff Bloom Operator* leads to a transition from the *Jammer Deployed State* in the *Do Decoy Missiles* problem space to the *Handled Missiles* state in the *Do EW Task* problem space.

Terminated: When the model is in the *Chaff Bloomed* state of the *Do Decoy Missiles* problem space.

7.5 Do Deploy Chaff Problem Space

This problem space is used to decoy a missile using chaff. The process involves working out a route to maximise the separation between the chaff cloud and the ship, then firing the chaff, and finally following the planned escape route. The basic state transition diagram showing the relationship

7.5.1 Calculate Escape Route Operator

Proposed: When in the *Initial State* of the *Do Deploy Chaff* problem space.

Applied: When in the *Initial State* of the *Do Deploy Chaff* problem space AND the operator is in slot on the state. The application of this operator leads to a transition from *Initial State* to *Escape Route Calculated* state. In addition, the planned new heading and planned new speed are added to the state.

Terminated: When the escape route has been calculated, such that the model is in the *Escape Route Calculated* state of the *Do Deploy Chaff* problem space.

7.5.2 Select Chaff Dispenser Operator

Select Chaff Dispenser is a mental operator, used by the operator to pick one of the four sets of chaff dispensers. One set of dispensers is located fore and aft on the port and starboard sides of the ship.

Proposed: When the model is in the *Escape Route Calculated* state of the *Do Deploy Chaff* problem space.

Applied: When the model is in the *Escape Route Calculated* state of the *Do Deploy Chaff* problem space AND the operator is in slot on the state. Application of this operator leads to a transition from the *Escape Route Calculated* state to the *Chaff Dispenser Selected* state. In addition, some substructure is added to the state to show the details of the chaff being deployed. The details of which chaff dispenser has been selected are added to this substructure.

Terminated: When the chaff dispenser has been selected, such that the model is in the *Chaff Dispenser Selected* state.

7.5.3 Select Chaff Rounds Operator

Proposed: When the model is in the *Chaff Dispenser Selected* state of the *Do Deploy Chaff* problem space.

Applied: When the model is in the *Chaff Dispenser Selected* state if the *Do Deploy Chaff* problem space AND the operator is in slot on the state AND there is some substructure on the state describing the details of the chaff being deployed. Application of this operator leads to a transition from the *Chaff Dispenser Selected* state to the *Chaff Rounds Selected* state. This operator makes use of external commands to select chaff rounds from the chosen chaff dispenser on the display. In order to do this the appropriate number of chaff objects have to be clicked on. The application of this operator also adds some substructure to the state to record the number of selected chaff rounds. Normally, only one chaff round is used.

Terminated: When the chaff rounds have been selected, such that the model is in the *Chaff Rounds Selected* state..

7.5.4 Arm Chaff Operator

Proposed: When the model is in the *Chaff Rounds Selected* state of the *Do Deploy Chaff* problem space.

Applied: When the model is in the *Chaff Rounds Selected* state of the *Do Deploy Chaff* problem space AND the operator is in slot on the state AND there is some substructure on the state showing the details of the chaff being deployed. Application of this operator leads to a transition from the *Chaff Rounds Selected* state to the *Chaff Rounds Armed* state. In addition, the status of the chaff (i.e., armed) is added to the chaff details substructure on the state. This operator makes use of external commands to click on the command button labelled "ARM" which forms part of the display screen

Terminated: When the selected chaff rounds have been armed, such that the model is in the *Chaff Rounds Armed* state.

7.5.5 Fire Chaff Operator

Proposed: When the model is in the *Chaff Rounds Armed* state of the *Do Deploy Chaff* problem space.

Applied: When the model is in the *Chaff Rounds Armed* state of the *Do Deploy Chaff* problem space AND the operator is in slot AND there is some substructure on the state describing the chaff being deployed. Application of this operator leads to a transition from the *Chaff Rounds Armed* state to the *Chaff Rounds Fired* state. In addition the status of the chaff (i.e., fired) is added to the chaff details substructure on the state. This operator makes use of external commands to click on the command button labelled "FIRE" which forms part of the display screen.

Terminated: When the selected chaff rounds have been fired, such that the model is in the *Chaff Rounds Fired* state.

7.5.6 Follow Escape Route Operator

Proposed: When the model is in the *Chaff Rounds Fired* state of the *Do Deploy Chaff* problem space.

Applied: When the model is in the *Chaff Rounds Fired* state of the *Do Deploy Chaff* problem space AND the operator is in slot on the state AND the desired speed has been calculated AND the desired heading has been calculated. Application of this operator leads to a transition from the *Chaff Rounds Fired* state to the *Following Escape Route* state. This operator makes use of external commands to enter the new speed and new heading values in the appropriate boxes on the display screen.

Terminated: When ship has started to follow the calculated escape route, such that the model is in the *Following Escape Route* state.

7.6 Do Deploy Jammer Problem Space

This problem space is used to decoy a missile using one of the two on-board jammers. The process involves selecting a missile to jam, then working out which jammer to use, and finally activating the jammer.

7.6.1 Select Jammer Operator

Select Jammer is a mental operator, which uses a rule of thumb to decide whether to use the port or the starboard jammer. It has two proposals, one for each of the jammers. The model is slightly simplified in that it uses the port side jammer by default.

Proposed: When the model is in the *Initial State* of the *Do Deploy Jammer* problem space.

Proposed: When the model is in the *Initial State* of the *Do Deploy Jammer* problem space AND the missile is located to the starboard side of the ship.

Applied: When the model is in the *Initial State* of the *Do Deploy Jammer* problem space AND the operator is in slot on the state AND the side on which the missile is located is recorded on the operator. Application of this operator leads to a transition from the *Initial State* to the *Jammer Selected* state. In addition, the jammer that has been selected is copied from the operator substructure to the state.

Terminated: When a jammer has been selected, such that the model is in the *Jammer Selected* state.

7.6.2 Request Jammer Permission Operator

Proposed: When the model is in the *Jammer Selected* state of the *Do Deploy Jammer* problem space.

Applied: When the model is in the *Jammer Selected* state of the *Do Deploy Jammer* problem space AND the operator is in slot on the state. Application of this operator leads to a transition from the *Jammer Selected* state to the *Jammer Permission Selected* state.

Terminated: When permission to use the jammer has been processed, such that the model is in the *Jammer Permission Requested* state.

7.6.3 Activate Jammer Operator

Proposed: When the model is in the *Jammer Permission Requested* state of the *Do Deploy Jammer* problem space.

Applied: When the model is in the *Jammer Permission Requested* state of the *Do Deploy Jammer* problem space AND the operator is in slot on the state AND the selected jammer is recorded on the state. Application of this operator leads to a transition from the *Jammer Permission Requested* state to the *Jammer Activated* state. In addition, the track of the missile being jammed is added to the state and the jammer status is set. This operator includes the use of external commands to enter the details of the track that is to be jammed on the display screen.

Terminated: When the jammer has been activated, such that the model is in the *Jammer Activated* state.

7.7 Do Track Helicopter Problem Space

The *Do Track Helicopter* problem space is used to perform the part of the EW Task which involves maintaining station with a friendly helicopter. This task involves keeping the helicopter at a distance of approximately 2 miles, and at a bearing of 90° from own ship.

7.7.1 Get Helicopter Details

Proposed: When the model is in the *Initial State* of the *Do Track Helicopter* problem space.

Applied: When the model is in the *Initial State* of the *Do Track Helicopter* problem space AND the operator is in slot on the state. Application of this operator leads to a transition from *Initial State* to the *Helicopter Details Known* state. In addition some substructure is added to the state to record the distance and bearing details for the helicopter. This operator make use of external commands to perceive the distance and bearing of the helicopter.

Terminated: When the model is in the *Helicopter Details Known* state.

7.7.2 Control Ship Heading Operator

Proposed: When the model is in the *Helicopter Details Known* state of the *Do Track Helicopter* problem space.

Applied: When the model is in the *Helicopter Details Known* state of the *Do Track Helicopter* problem space AND the operator is in slot on the state AND the helicopter details are recorded on the state. Application of this operator leads to a transition from the *Helicopter Details Known* state to the *Ship Heading Controlled* state. In the final version of the model, this operator will make use of external commands to change the heading of the ship as appropriate using the display screen.

Terminated: When the new ship heading has been set, such that the model is in the *Ship Heading Controlled* state.

7.7.3 Control Ship Distance Operator

Proposed: When the model is in the *Ship Heading Controlled* state of the *Do Track Helicopter* problem space.

Applied: When the model is in the *Ship Heading Controlled* state of the *Do Track Helicopter* problem space AND the operator is in slot on the state AND the model is in the *Handled Missiles* state of the *Do EW Task* problem space. Application of this operator leads to a transition from the *Ship Heading Controlled* state to the *Ship Distance Controlled* state in the *Do Track Helicopter* problem space. It also leads to a transition from the *Handled Missiles* state to the *Initial State* of the *Do EW Task* problem space.

Terminated: When the distance between the ship and the helicopter has been successfully controlled, such that the model is in the *Ship Distance Controlled* problem space..

7.8 Do Control Ship Heading Problem Space

The *Do Control Ship Heading* problem space has not been implemented in the initial version of the model. It is described here for the sake of completeness, and to provide a basis for future implementation.

7.8.1 Get Helicopter Bearing Operator

Proposed: When in the *Initial State* of the *Do Control Ship Heading* problem space.

Applied: When in the *Initial State* of the *Do Control Ship Heading* problem space AND the operator is in slot on the state. Application of this operator leads to a transition from the *Initial State* to the *Got Helicopter Bearing* state. This operator uses external commands to estimate the bearing of the helicopter with respect to own ship from their relative positions on the display screen.

Terminated: When the bearing of the helicopter has been estimated, such that the model is in the *Got Helicopter Bearing* state.

7.8.2 Read Ship Heading Operator

Proposed: When the model is in the *Got Helicopter Bearing* state of the *Do Control Ship Heading* problem space.

Applied: When the model is in the *Got Helicopter Bearing* state of the *Do Control Ship Heading* problem space AND the operator is in slot on the state. Application of this operator leads to a transition from the *Got Helicopter Bearing* state to the *Got Ship Heading* state. This operator makes use of external commands to read the ship heading from the display screen.

Terminated: When the ship heading has been read from the display, such that the model is in the *Got Ship Heading* state.

7.8.3 Change Ship Heading Operator

Proposed: When the model is in the *Got Ship Heading* state of the *Do Control Ship Heading* problem space.

Applied: When the model is in the *Got Ship Heading* state of the *Do Control Ship Heading* problem

the *Got Ship Heading* state to the *Ship Heading Altered* state. This operator makes use of external commands to enter a new value for the ship heading on the display screen.

Terminated: When the new ship heading value has been entered, such that the model is in the *Ship Heading Altered* state.

7.9 Do Control Ship Distance Problem Space

The *Do Control Ship Heading* problem space has not been implemented in the initial version of the model. It is described here for completeness, and to provide a basis for future implementation. The ship separation provides the basis for determining whether the speed of the ship should be increased or decreased. The basic state transition diagram showing the relationship between the operators and the states is shown in Figure 9.

7.9.1 Estimate Ship Separation Operator

Proposed: When the model is in the *Initial State* of the *Control Ship Distance* problem space.

Applied: When the model is in the *Initial State* of the *Control Ship Distance* problem space AND the operator is in slot on the state. Application of this operator leads to a transition from the *Initial State* to the *Ship Separation Estimated* state. This operator makes use of external commands to judge the relative distance between own ship and the helicopter.

Terminated: When the model has worked out the separation between own ship and the helicopter, such that the model is in the *Ship Separation Estimated* state.

7.9.2 Increase Ship Speed Operator

There are two cases where the ship's speed needs to be increased. The first is when the helicopter has moved too far ahead of the ship in a northerly direction; the second is when the helicopter is too far away from the ship in a lateral direction (either east or west). There is an operator proposal to deal with each case.

Proposed: When the model is in the *Ship Separation Estimated* state of the *Control Ship Distance* problem space AND the helicopter is too far ahead of the ship.

Proposed: When the model is in the *Ship Separation Estimated* state of the *Control Ship Distance* problem space AND the helicopter is laterally too far away from the ship.

Applied: When the model is in the *Ship Separation Estimated* state of the *Control Ship Distance* problem space AND the helicopter is too far ahead of the ship AND the operator is in slot on the state. Application of this operator leads to a transition from the *Ship Separation Estimated* state to the *Ship Speed Altered* state.

Terminated: When the ship's speed has been updated, such that the model is in the *Ship Speed Altered* state.

7.9.3 Reduce Ship Speed Operator

Proposed: When the model is in the *Ship Separation Estimated* state of the *Control Ship Distance* problem space.

Applied: When the model is in the *Ship Separation Estimated* state of the *Control Ship Distance* problem space AND the operator is in slot on the state AND the ship is too far ahead of the helicopter. Application of this operator leads to a transition from the *Ship Separation Estimated* state to the *Ship Speed Altered* state.

Terminated: When the ship's speed has been updated, such that the model is in the *Ship Speed Altered* state.

8. References

Annett, J., Duncan, K. D., Stammers, R. B., and Gray, M. J. (1971) *Task Analysis*. Training Information Paper No. 6. London, UK: HMSO.

Bass, E.J. & Baxter, G.D. (1995) *Proposal for Initial "Eye Soar"* Working Paper WP/R3BAIA0005/004, Dept. of Psychology, University of Nottingham.

Baxter, G.D. & Ritter, F.E. (1996a) *Designing Abstract Visual Perceptual and Motor Capabilities for Use by Cognitive Models*. Technical Report No. 36, Centre for Research in Development, Instruction and Training, University of Nottingham.

Baxter, G.D., & Ritter, F.E. (1996b) Model-computer interaction: Closing the action perception loop for cognitive models. In *Proceedings of the First International Conference on Engineering Psychology and Cognitive Ergonomics*, Stratford-upon-Avon, UK.

Chapman,T., Ritter, F.E. & Baumann, M. (1996) *Electronic Warfare Task Manual* Working Paper WP/R3BAIA0005/013, Dept. of Psychology, University of Nottingham.

Booth, S. & Sheppard, C. (1996) *Synthetic CIS Tasks for SOAR Modelling of Rapid Decision Making in CIS: EW Task*. Working Paper DRA/CHS/HS3/WP95/005/1. Farnborough, UK: DERA.

Boy, G. (1990). *Intelligent Assistant Systems*. London, UK: Academic Press.

Card, S., Moran, T. & Newell, A. (1980)

Congdon, C.B. & Laird, J.E. (1997) *The Soar User's Manual Version 7*. Electrical Engineering and Computer Science Department, University of Michigan.

Congdon, C. B. & Schwamb, K.B. (1997) *The Soar Advanced Applications Manual*. Electrical Engineering and Computer Science Department, University of Michigan.

Diaper, D. (1989) *Task Analysis for Human-Computer Interaction*. Chichester, UK: Ellis Horwood.

Laird, J.E., Newell, A., & Rosenbloom, P.S. (1987) Soar: An architecture for general intelligence. *Artificial Intelligence*, 18, 87-127.

Norman, D.A. (1988) *The Psychology of Everyday Things*. New York, NY: Basic Books.

Ong, R. & Ritter, F.E. (1995) Mechanisms for routinely tying cognitive models to interactive simulations. In *HCI International '95*. Osaka, Japan: July, 1995.

Patrick, J. (1992) *Training: Research & Practice*. London, UK: Academic Press.

Ramsay, A. (1995) *OOPSDG Modelling Environment for the Centre for Human Sciences*. Technical Report No. DRA/CIS(SS5)/1026/9/2. Portsdown, UK: DRA.

Rassouli, J. (1995) *Steps towards a process model of mouse-based interaction.* MSc Thesis, U. of Nottingham.

Sherrill-Lubinski Corporation (1994) *SL-GMS Technical Overview*. Corte Madera, CA: Sherrill Lubinski Corporation.

Wherry, R.J. (1976) The human operator simulator—HOS. In T. B. Sheridan & G. Johannsen (eds.), *Monitoring Behavior and Supervisory Control*. New York, NY: Plenum Press.

Yost, G.R. (1996). Implementing the Sisyphus-93 task using Soar/TAQL, *International Journal of Human-Computer Studies*, 44, 281-301.

9. Appendix A: The Eye/Hand Extension

The EW Task model has an extension added to it which allows the model to use the task application interface in the same sort of way that a real user would. This extension provides the model with a visual perceptual capability, and a motor action capability. These allow the model to perceive what is displayed on the screen at any particular moment, and to make changes to the system using the keyboard and mouse, as appropriate.

The extensions, which build on earlier attempts in this area (Bass & Baxter, 1995; Rassouli, 1995), reside in the application interface and are implemented in SL-GMS (Sherrill-Lubinski Corporation, 1994). The Soar model manipulates the eye and the hand by using external operators. Communication between the model and the extensions is realised using a socket-based utility, MONGSU (Ong & Ritter, 1994).

Each of the external operators is described below, in order to illustrate how they can be used by the model to interact with the application system. Thus, for each of the operators, the conditions needed to apply the operator are described, together with the actions that result from their application. Since the extensions are implemented as operators, the changes that they make to the internal state of the operator are all permanent, in that they can only be removed explicitly by another operator, for example.

The operators used to interact with the application interface are:

- Saccade
- Fixate
- Mouse move
- Hand-eye co-ordination

Saccade Operator

Fixate Operator

Move Mouse Operator

Need the command grammar here too

10. Appendix B: Knowledge Elicitation

A knowledge elicitation exercise was carried out to supplement the task analysis for the EW Task. This consisted of two parts:

1. Video taping an expert as he performed two of the scenarios used in the EW Task experiment (scenarios 2 and 11). The video protocols of the expert doing the task will be used as a source of expert knowledge, and will also serve as reference points for comparison between how the expert does the task, and the off-line description of how to do the task.

2. Semi-structured interviews with the same expert, which are described in more detail below.

A semi-structured interview technique was selected because the task is rather artificial, although it does have some surface similarity with real tasks. It was regarded as important to be aware of the limitations of the EW Task, and explicitly document them. In addition, refinement of the initial task analysis, and discussions with DERA which had provided a framework for the development of the model, also raised a number of specific points which needed to be addressed or resolved. Most of these points related to strategic matters, such as how to deal with multiple missiles; performance under single missile conditions appears relatively straightforward. The interview was therefore based around gaining confirmatory evidence for the strategy proposed for handling single missiles, and answering particular questions for more complex scenarios.

The interviews were centred around talking through two scenarios: one where only single missiles were involved, and one where multiple missiles were involved. The interviews were not recorded, since the expert was being videotaped separately whilst he performed the EW Task. A number of questions were drawn up prior to the interview, as a checklist to make sure that all of the identified points were covered. The benefit of the semi-structured interview is that it also allows the expert to raise other points which can be captured and investigated on the fly.

A second interview was performed as a follow up to the initial interview and the videotaping. In some cases there were particular elements of the expert's task performance which emerged from looking at the videotape, which had not been considered beforehand. Some of the questions used to impose some structure on the second interview also arose from the implementation of the model. There were some particular aspects of the task, which required more detail, in order to clarify the strategy used to perform individual sub-tasks.

The transcripts from the two interviews have been merged together, in order to provide a coherent discourse about how to do the EW Task.

10.1 Interview Preamble

The initial scenario is a relatively straightforward one. The aim is to try and prevent own ship from being hit by a single incoming missile, whilst at the same time trying to maintain station with a friendly helicopter. The expert was asked to describe how he would accomplish this task. He was informed that he could be prompted with questions at various points throughout the description in order to clarify issues which may be unclear, or require further description.

The second scenario involves a number of missiles arriving within a relatively short space of time. Again, the aim is to try and decoy the missiles, whilst maintaining station with the friendly helicopter. Some of the questions for this scenario are similar to those which were asked about the single missile scenario, but they were designed to try and pick up any differences in strategy that may be present.

10.2 Questions used to structure the Knowledge Elicitation

The following questions were used to provide a loose framework for making sure that the interviews covered all of the appropriate points which had been identified beforehand. Where appropriate, explanatory comments have been appended, and shortened versions of the answers provided by the expert are shown in italics.

1. What strategy would you use to decoy a single radar detection missile, coming in at a fixed speed?

(This is answered at length in the transcript below.)

2. How would you choose between using chaff and using one of the jammers?

If both are available, use both.

3. In a single missile scenario, when would you launch chaff? (This is to check whether distance or time is used, and to see whether missile speed is taken account of.)

Chaff is launched as early as possible.

4. What strategy is used for the deployment of chaff?

Always use it, as early as possible.

5. Do you check to see if chaff has bloomed before moving off?

Not necessarily before moving off, but always check for blooming.

6. After firing chaff, what changes would you make to the speed of own ship?

Generally ramp up to maximum speed.

7. After firing chaff, what changes would you make to the heading of own ship?

Could not quantify the change – the critical factor is to get a blast of relative wind.

8. When would you zoom in and out of the PPI?

Zoom in to keep the missile at the edge of the display. Zoom out after decoying a missile to check for others. Zoom in to regain station with the helicopter.

9. When would you use the small PPI?

Never.

10. Do you check to see that a missile is successfully decoyed?

Yes. Look for a change in the bearing of the missile to indicate deflection.

11. What strategy would you use to decoy several missiles, all of the same type, all coming in at fixed speeds from different directions at the same time?

Always select the one which will arrive first to decoy first.

12. How do you decide when to look for missiles, and when to stop looking for missiles?

Always look for missiles. If there is enough time, try and maintain station with the helicopter.

13. Do missiles tend to appear in bunches?

Yes, and a concerted attack may come from separate directions.

14. Do you try and group missiles together, or do you always deal with them one by one? If you *do* group them together, what criteria do you use to determine the groupings?

Group them by bearing, and estimated time of arrival.

15. Do you handle missiles which are not heading towards own ship?

In the real world, yes. In the experiment it does not matter.

16. How do you select which missile to decoy?

The one which will arrive first.

17. When do you turn the jammer off?

When the jammed missile approaches 10 miles. Can also be turned off when changing heading.

18. Does having the jammer turned on affect your strategy for using chaff? (Turning the ship may mean losing track of the jammed missile.)

No, apart from the effect on heading.

19. Is there a different strategy for breaking lock when a missile locks on?

Yes, see below.

20. Do you decoy all missiles before looking to see if any others have been identified?

No, time to contact is most important feature.

21. How do you know which missiles have been decoyed? (Are they tagged in some way? More specifically how do you get round the problem of a missile being tagged as decoyed, and then whilst dealing with another missile you head off into the path of the previously decoyed one!)

A shift in bearing towards the chaff indicates a successful decoy.

22. Once a missile has been decoyed do you ever return to check it? (Suppose, for instance that you tag as missile as decoyed (with the jammer) and you turn the ship which puts the missile outside the arc of the jammer. If you do not revisit that missile you cannot mark it as undecoyed.)

Missiles are checked until a shift in bearing (towards the chaff) is detected.

23. Do you plan an escape route first, then fire chaff, and then follow the planned route?

A qualitative approach is used: pick an approximate heading, fire chaff, and do it.

24. Is it an intensive task? In other words is it a case of you get called to station and then sit and watch the screen and react until told to stand down?

As it stands it is not a very intensive task for an expert, but it is in the real world, in bursts.

25. How important is relative wind? (It appears to be unimportant when a decision has been made to use the jammer.)

Relative wind is used to work out an approximate heading.

26. Do you feel that you actually plan, or are things based more on gut-feelings?

Based on planning, training, briefing, and years of experience.

27. What are your feelings about the usefulness of the widgeon as a means of helping to calculate escape routes? (This is not strictly part of the EW Task, but it does provide an insight into the strategy used to determine escape route i.e. calculation or more qualitative.)

Excellent as a training device.

28. How important is time?

Critical.

29. How do you calculate the refinement to the heading and speed of own ship?

It is usually a 2-3° refinement at most to make sure that both 909s can be used.

30. How often do you check the relative wind value?

Very often, both the pointer and the value.

31. Do you try to use existing chaff as part of the decoy process?

Yes.

32. What is the approximate failure rate of chaff?

5%.

33. When checking the missiles, how do you determine what is new and what is old?

Still check them all, reacting to the oldest wave first.

34. Do you keep track of what the jammer is doing *all* the time?

Yes, due to the home-in-on-jam capability of the missiles.

35. When do you look for a deflection in bearing?

Constantly, although it only becomes detectable on shorter range PPI display.

36. How is the deflection in bearing indicated?

By a deviation in the track on the display, which bends towards the chaff.

37. How do you decide what to do to regain station with the helicopter?

Try and maintain speed, so circle round if own ship heading is already changing.

38. When trying to regain station, does it become the main focus of the task?

10.3 Transcript of the Interviews

The task was assessed as having a relatively high level of realism. When asked to rate it on a scale of 1 to 10, the expert said he would give it around 7.5 to 8. The main drawback with the simulation, however, is that the task can easily become too routine. So, for a domain expert, it can be learned in a fairly short space of time. The expert pointed out that he had attained his level of expertise through a lifetime of working in the domain, attending and giving training courses, making sure that he kept abreast of current developments in the field, and actually doing the task for real.

He conceived of the task as essentially a scanning task, with the real part of his job being to protect own ship, and only revisited the need to maintain station with the helicopter when he had nothing else to do. As far as he was concerned the main task was simply to protect own ship. When it came to maintaining station with the helicopter, however, that became the main focus of the task.

Maintaining station with the helicopter involves trying to keep the speed of own ship at a relatively high level, rather than sitting dead in the water. This is largely due to the length of time, and energy required to speed up and slow down the ship. It was noted that the expert often circled round to regain station, although he pointed out that this usually happened when he was already turning, when he started focusing on the task of regaining station. The expert also pointed out that it is rare for ships to maintain a course for more than a short space of time.

Initially he would use the 100 mile range on the PPI, and scan for planes. When a threat comes in and launches a missile his first reaction is to launch chaff, then increase to maximum speed and change heading. The change of heading is a qualitative one, rather than an exact heading, he looks for an approximate direction. The aim is to try and buy as much time as possible, and refine the course later, as appropriate. He said that he could not quantify the calculation of the change of heading, although the critical factor was to get a good blast of relative wind. The change in heading may be non-optimal, in order to make sure that the on-board countermeasures, both fore, and aft, could be deployed. In other words, the ship may present a radar signature slightly larger than the minimum. The refinement to the course is often decided by the soft kill manager, but is usually restricted to 2-3°. It was also noted that in rough conditions the ship's heading can be subject to variations up to 15°.

Generally he would launch just one round of chaff per missile, although he admitted that during the videotaping sessions he had launched two after he had been hit. After launching the chaff, he also revisited the chaff to make sure that it had bloomed, which he said was fairly apparent from the display. If it failed to bloom, which happens in about 5% of cases, he would launch another. His basic strategy is to launch chaff early, rather than wait until the missile reaches some particular distance from own ship. If there was already chaff in existence which could be of use against an incoming missile, however, it would be utilised, although this strategy requires that the chaff bloom be carefully watched, since it would disappear in a shorter space of time than a new bloom, and may therefore need to be reseeded.

He tracks missiles by zooming in and out of the PPI. The aim is to try and keep the missile on the edge of the scope. He can tell when a missile is decoyed by a drift in its bearing, so using the minimum range of the PPI which still allows the missile to be displayed makes the bearing drift easier to see. In the EW Task, the *only* way to detect that the missile has been drawn by the chaff is by a change in the displayed track, and this becomes more apparent when the range of the PPI is set to a relatively low value.

The expert did not use the small PPI display once he realised that he could not alter the scale of it. He said that if he had been able to, he would have set the range on the small PPI to be 100 miles, so that he could always have an overall picture of what was happening around him.

Once a missile passes the ship, he no longer pays any attention to it. In addition, once he notices an appropriate drift in bearing of a missile (towards the chaff cloud) he no longer pays any attention to it. In reality, there are several pairs of eyes trained on incoming missiles, so if a missile switched back to track own ship again, someone would detect it, and draw attention to it.

His strategy is to always ask for the jammer, with the idea being to use all means available to decoy an incoming missile. He would use both chaff and the appropriate jammer, rather than selecting one over the other. When the jammer is used, however, it has to be turned off when the missile reaches a distance of around 10 miles from own ship. This is because the missiles have a home on jam capability which comes into play at this distance.

The status of the jammer has to be continually checked, rather than simply checking it when there is a new missile to decoy. The reason for this is the home in on jam capability of the missiles. If the jammer is left unchecked, it increases the chance of a missile locking onto the jammer at a relatively short range. This explains why the jammer status is checked before asking for permission to use it.

After a single missile attack he would zoom out to the 100 mile range again on the PPI, and then if there were no other missiles, he would zoom in again to regain station with the helicopter.

He always concentrated on the relative wind. His aim was to try and generate an effect that created a lot of relative wind which would help to maximise separation of the chaff and own ship. Rather than the precise direction of the relative wind, however, he was mainly concerned about getting "pockets full" of wind. This explained why he would do a coarse adjustment to ship's heading first, which he would refine later, as required.

The application interface for the task provides two relative wind indications. There is a numeric display which shows the relative wind heading and speed, and there is a pointer on the PPI which indicates the direction and speed (by the length of the pointer) of relative wind. The relative wind values were checked frequently, because in relatively calm conditions, the pointer can swing around on a frequent basis, thus making it harder to determine the best current relative wind.

When it came to choosing a chaff dispenser, he said that he would choose the dispensers that were closest to the threat after a coarse heading had been selected. He pointed out that although he would usually set the ship speed to maximum, on a real ship he would also have to take into account the size of the change in heading. The reason for this is that when a real ship is turning at full speed, it heels during manoeuvring, which means that the chaff dispensers can end up firing into the water.

Under multiple missile scenarios, missiles would tend to come in waves, rather than in ones or twos. The attacks could also be part of a combined attack from several directions.

The strategy for dealing with multiple missile scenarios is to react to a single threat, then move onto the next, although threats with the same heading, and same estimated time to contact could be grouped, and decoyed together. Once a threat had been dealt with it was forgotten almost immediately. Otherwise the general strategy is to try and give some time to the next most immediate threat, and keep cycling round the threats which have not been successfully decoyed, actively ignoring them when an appropriate change in heading is detected, indicating that the missile was heading for the decoy. The expert agreed that it was very much like plate spinning, where you give most attention to the plates that need it most, and can let others sit for a while.

If a threat was not heading for own ship, the general strategy in the real life situation is still to try and deal with it. The reason for this is that ships usually travel in convoys, so you do what you can to help each other. On the single ship scenarios used in the model, however, dealing with missiles not heading towards own ship was not seen as a problem.

The strategy used to select which missile to deal with first is based on the estimated time to contact with own ship, rather than simply picking the closest one. Again the expert reiterated the key issue of needing to buy time wherever possible.

With multiple missiles, the jammer would be deployed against radar radiation missiles as a priority. The home on jam issue is also important, so the jammer needs to be deactivated at a distance of about 10 miles. The use of the jammer influences how changes in heading are carried out, due to the restricted coverage of the jammer. If necessary, the jammer would be switched off before setting a

So for example, if a missile was deemed to be decoyed (as detected by a change in bearing) and it was being jammed, then the jammer would be deactivated.

If a missile becomes locked onto own ship the strategy to break lock involves making a large target to decoy the missile. A check would be made on the deployed chaff to ensure that it was still blooming. As the chaff faded, more would be fired to enlarge the signature of the chaff cloud.

The task was regarded as being fairly steady by the expert, based on his experiences of the two scenarios he had encountered. He felt that it could be difficult for people without the appropriate background. He also pointed out that in the real world, the loading of the task is somewhat different. He agreed that it was somewhat like industrial process control, which has been described as months of tedium, punctuated by minutes of sheer panic, when things go wrong.

He felt that his approach to doing the task was based to a large degree on planning. Before going into such a situation he would try to be as fully briefed as possible, so that he would have a high level of expectation about what could happen. When something did happen he would then be able to react by calling up the appropriate plan automatically, rather than have to plan on the spot.

When asked about the importance of time in the task, the expert deliberated for a while, before deciding that, rather than time, relative speed was the critical factor. He would pay attention to the relative ground speed of the missiles, and then project ahead on that basis, to decide which actions were appropriate.

Finally the expert was asked about the utility of the widgeon. He felt that it was an excellent training aid, but would be of limited use in real situations. His general strategy involves an initial qualitative plan, moving in a general direction, based on creating a blast of relative wind. Using the widgeon may help to generate a more precise heading, but would also mean getting bogged down in calculations, when the critical factor is to try and buy time, in any way possible. The expert also pointed out that, on reflection, the display of the ship's countermeasures reminded him of the widgeon shape, and could be used to help in calculating new courses.

The picture of the ship with its on-board countermeasures was not regarded as distracting even though it always pointed in the same direction. The expert said that he always mentally rotated it to fit on with the current heading of own ship, so that he would be immediately aware of which jammer or chaff dispenser would be appropriate.

10.4 Post Interview

There are a number of inferences which can be drawn on the basis of the Knowledge Elicitation and the Videotapes of the expert. These facts were all taken into account, and used to update the three levels of model and refine the analysis of the task.

11. Appendix C: Critiquing the Approach

There are no recommended methods for developing cognitive models from scratch. Most of the available tools and techniques that may be used come from the fields of Software Engineering or Artificial Intelligence. In many cases, the development of a model is done in an ad hoc fashion, which makes it difficult to see where certain parts of the model come from, and why certain parts of the task are done in particular ways. Yost (1996) is an exception (or possibly *the* exception) to this rule, although much of the important detail is lacking from that article too.

This appendix critically examines the approach described in this document. The intention is to compare the approach presented here with that of Yost, and to highlight areas that require careful attention during the development of cognitive models. It also considers some possible alternative methods that could be utilised in the development of cognitive models, especially in Soar.

There are no coherent methods or toolkits which can be used to develop all three levels of models (Knowledge Level, PSCM, and SLCM). More generally, the analysis of complex systems is not an easy task, and requires the analyst to improvise in an intelligent manner (Patrick, 1992). The approach that was used here involves improvising by using existing tools and techniques, mainly from Software Engineering domain, as a means for developing cognitive models.

11.1 Defining the Knowledge Level Model

11.1.1 Critique of HTA

There are a number of problems with all systems analysis methods, and HTA is no different in that respect. The biggest problem is that the resultant analysis is necessarily prescriptive to an extent that varies with the number of novel components involved in the task. In addition it is very difficult to be able to show the completeness of the analysis. One of the main advantages of HTA is that it is relatively easy to use, but to use it well requires practice and experience. Some of the general problems of the available methods can be overcome by ensuring that the results of the analysis are subject to scrutiny by a number of people who are familiar with the task.

On its own, however, HTA does not appear to be enough. Some problems were identified after the initial HTA had been drawn up and scrutinised. In particular, there were some omissions, and some confusions over details of strategies. The Knowledge Elicitation exercise described in Appendix B helped to mitigate these problems, and is recommended as an adjunct to any other methods used to develop cognitive models.

11.1.2 Object-oriented Approach

One way of testing the analysis of the model for completeness involves taking account of the various bits of task information which are presented on the computer display screen. For each of these consideration is given to where it comes from, and how it can be used and altered. In this way it should be possible to increase confidence in the coverage of the model, since there should be Soar mechanisms (operators or elaborations) to implement the operations performed on the information (knowledge) required to do the task. This general idea bears some similarity to the ideas of object oriented development.

11.1.3 Other Possible Methods

Potentially, it appears as though most software structured development methods have something to offer to cognitive model development. In particular, those methods which draw a distinction between data and operations seem most appropriate. The most obvious candidate is Jackson Structured Design. There appears to be some compatibility between JSD and Soar models. JSD emphasises the

importance of hierarchical data structures which in this case can be mirrored by knowledge structures. In addition, JSD also focuses on the attachment of operations to data structures, at all levels in the hierarchy, which resembles the notion of having operators act on the task knowledge.

It would also be interesting to examine the possibility of using the Knowledge Blocks method (Boy, 1990) of representation as a way of developing models. This method was inspired by the work on Soar so there should be some affinity between it and Soar. It centres around the idea that procedures are represented as blocks which contain a description of goals, a set of conditions, and a set of actions which can be used to achieve the goals. The conditions are subdivided into triggering preconditions, abnormal conditions, and contextual conditions.

11.2 Defining the Problem Space Computational Model

There is a predefined notation for representing problem spaces and operators which Yost (1996) used. The particular representation is peculiar to Soar, however, with problem spaces being represented by triangles, and the operators which lead to transitions between problem spaces being shown as labels on directed arrows between the problem spaces. The effects of applying the operators are not shown in this representation.

Although State Transition Diagrams (STDs) are shown in the description of the Symbol Level Computational Model (Section 7) they are really part of the definition of the PSCM. Using some form of State Transition Diagram (STD) has a number of advantages. Perhaps the most obvious one is that they are relatively widely used, and so there is a reasonable likelihood that cognitive modellers should be able to understand them with little or no effort. The other main advantage is that they inherently focus attention on states within the model, which is particularly important for Soar models at least, since states (or perhaps more correctly the problem space/state combination) are one of the main constructs within Soar. Using STDs also provides a way of graphically illustrating the model's behaviour before it is implemented.

The use of STDs in modelling the EW Officer's task emphasises just how simplistic the model really is. Each of the problem spaces is implemented as a linear sequence of operator applications, which take the model through a fixed sequence of states. In reality the ordering of the transitions between states is much less clear cut, and will depend on factors such as time, stress, breadth of experience and so on.

If STDs *are* adopted as a way of defining the Problem Space Computational Model, it becomes possible to utilise existing tools that support the use of STDs, such as some CASE tools, for example. One big advantage of having automated support for STDs is that it imposes a level of rigour and consistency that is difficult to enforce when everything is done by hand, During the documentation of the EW Officer's Task model, for example, a relatively large amount of time was spent checking to make sure that the diagrams and the text accompanying them matched the Soar code.

11.3 Defining the Soar Implementation Model

The only technique that has been adopted in the definition of the Soar Model is the use of a very loosely structured pseudo-English. This is used to describe the conditions and actions of the productions which propose, apply and terminate operators. Although a more formal mechanism could be employed, it is unclear how useful it would be, since Soar productions are essentially free format and there are no keywords in the Soar programming language. It would, however, be useful to be able to generate the list of operators and states automatically from the STDs produced during the development of the PSCM.

12. Appendix D: Data/Knowledge Structures

The structure of the state is actually made up of a composition of substructures. In this section the substructures are identified, with all of the attributes, and the range of possible values for those attributes.

Ship	current course	0..359
	current speed	0..30 (no limit)
	new course	0..359
	new speed	0..30 (no limit)
	port jammer	selected..unselected
	starboard jammer	selected..unselected
	chaff dispenser	port fore, port aft, starboard fore, starboard aft
	chaff selected	yes..no
	chaff armed	yes..no
	chaff fired	yes..no
Missile	bearing	0-359 (relative to ship)
	speed	???
	locked-on	yes..no
	distance	???
	jammed	yes..no
Helicopter	bearing	0-359 (relative to ship)
	speed	???
	distance	???

Caveats, problems, thoughts, suggestions, etc.

The initial model will be limited to single missile scenarios. Multiple missile scenarios will involve different strategies, depending on the number and type of missiles and their headings.

For the expert there should be a different set of perceptual cues used, according to the literature. This comes down to mental models of the task, and the fact that they are qualitatively different for novices and experts.

Control issues i.e. how do the various bits of the task analysis really fit together?

How to decide what's an operator?

What gets put on the state by operators?

The eye-hand model. Where does this get laid out? Really, the stopping condition for the task analysis is the point where calls are made to eye and/or hand "functions". What sort of changes do the eye hand functions make to the state–are they operator changes, or are they elaborations? They could be either: one suggests that you simply forget about certain things, and then load up all the new knowledge again, the other suggests that you just update values. I would suspect that the former view is more correct, since you "tag" an instance of an object so that you can recognise it as "that" object, rather than have a generic object. Partial schemata in Soar? Has anybody done this sort of work?